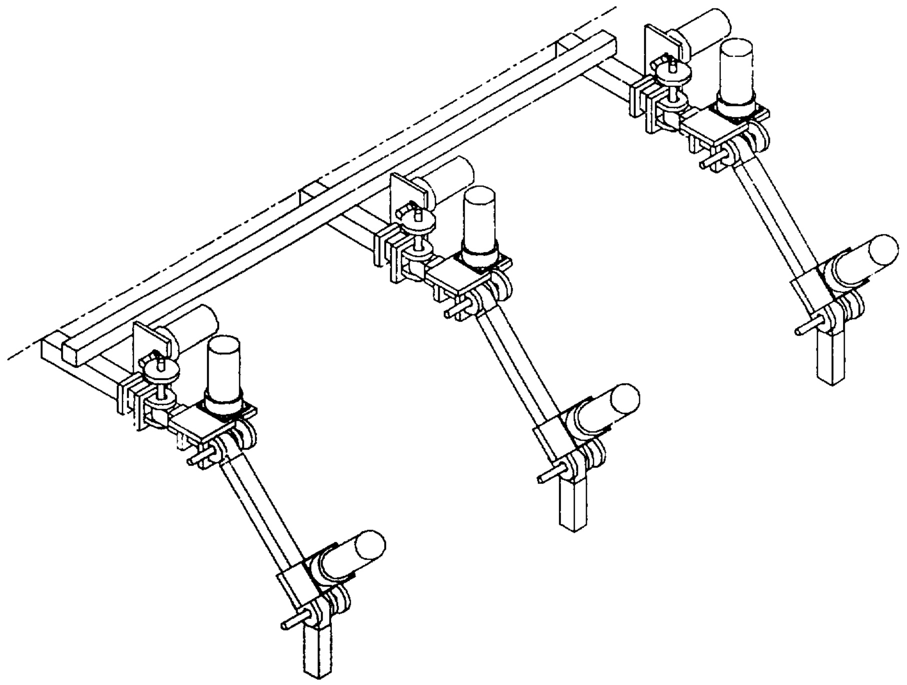
HUMAN-LIKE ROBOTS for SPACE AND HAZARDOUS ENVIRONMENTS

NASA/USRA Advanced Design Team
Departments of Mechanical Engineering
Computer Science
Electrical Engineering
and Psychology



Allen Cogley, David Gustafson, Warren White, Ruth Dyer
Edited by: Tom Hampton and Jon Freise

Table of Contents

List of Figures

List of Tables

Part I

Project Description

1 Goals

1.1 Three Year

The three year goal for this NASA Senior Design team is to design and build a walking autonomous robotic rover. The rover should be capable of rough terrain crossing, traversing human made obstacles (such as stairs and doors), and moving through human and robot occupied spaces without collision. The rover is also to evidence considerable decision making ability, navigation and path planning skills.

When began this project had the title "Human like Robots in Space and Hazardous Environments". Within this scope the design team was free to choose what it felt the projects goals should be. The first group of criteria the design team picked was that the project should be some form of mobile robot, that it be as autonomous and intelligent as possible, and that some form of the robot actually be constructed. These goals were chosen as being the most interesting to the group as a whole. Several variants of these ideas were discussed:

Hazardous waste cleanup: The chernobyl accident demonstrated several instances where remotely operated or robotically controlled earthmovers would have been useful (or life saving). This idea was discarded after noting that the large equipment needed was far beyond our budget.

Planetary Rover: The ability to range widely and collect samples, data, and pictures would be of benefit for any future space exploration missions. A mobile robot was envisioned that could traverse rough terrain, carrying a variety of sensors and instruments, with enough intelligence to travel without a human operator.

Hazardous waste site scout: A mobile robot that could move freely about a disaster site, allowing human operators to assess the damage without endangering themselves was another suggested idea.

The last two ideas were combined into the current three year goal when it was noted that the same abilities to cross rough terrain on other planets would allow a rover to explore hazardous waste sites. It was further noted that if the rover could also open doors, and climb stairs it would be able to access almost all indoor areas that humans could reach, and those abilities became part of the goal. The choice for a legged rover design seemed best for maximizing rough terrain crossing ability. The decision for supporting an autonomous robot as opposed to a teleoperated design was to allow a rover to operate at great distances from Earth, without hindrance of lag in communications. The high intelligence factor was also supported as it would allow an operator to control several rovers.

1.2 First year

The job of the design team the first year was to pick a project goal, and then attempt to define just what needed to be done to reach that goal. It became apparent rather quickly that the design group lacked practical experience building robots, and without that experience any design done would be flawed and fail. So the first year goal was set as building a working prototype of the walking robot rover for the purpose of learning as much as possible about mobile robot design.

The main criteria for the prototype were as follows:

Six legs: This was to begin the exploration into legged robot design. The choice of six legs allowed for a stable walking platform and a simpler overall control.

Three joints per leg: Three joints are the minimum needed for the robot to walk without its feet sliding or slipping.

On Board Intelligence: By restricting all processors to be on board the robot, the prototype would force realistic control schemes to be investigated.

Off Board Power: It was soon discovered that the cost of high efficiency electric motors needed to operate on battery power were out of our price range.

Stair Climbing: The prototype needed some target obstacle to test its terrain handling abilities. Stairs are a readily available obstacle in the indoor laboratory environment, and one of the needed criteria for meeting the three year goal.

The prototype was also to be designed in a modular way which allowed simple assembly and reconstruction. This requirement was to allow easy testing of various joint and leg designs, and terrain crossing abilities.

2 Implementation

The bulk of this report details the efforts of the design team to accomplish the first year goal of a walking six legged robot. While each of the design teams major groups, chassis, electronics, and controls (see project organization) have each written detailed sections to explain how the prototype was implemented, here is a feel for the scope of the robot. The robot measures over 40 inches long, has a leg span of 3 feet and weighs over 60 pounds. It has three joints per leg (as specified by the goals) and one electric motor and gear train drives each joint. Modularity of design has been stressed, to the point that 10% of the robot's weight is dedicated to nuts, bolts and other fasteners. There are a total of nine microprocessors used in the control of the robot. Three of those processors will be mounted on the central body and will run the higher level behavior based motion planning software. A graphical simulation package has been written to test this software before hardware integration. The other six processors are to be mounted out on the legs of the robot to control the leg motors and gather data from a vast array of sensors. Those sensors include two types of position feedback on each joint, foot force feed back, and two types of collision detection. Again modularity has been stressed in the electronic design, so that the only connections running from the robots main body out to each leg (and through the joints) are motor power, logical power and a single serial cable. While the dozens of pages of schematic drawings, blueprints and pages of code are testimony to the amount of work that has been done so far, the NASA team realizes it has just begun to learn what is needed to reach the three year goal of a fully autonomous walking robot rover.

3 Project Organization

At the end of the second semester the design team had over 20 members, 7 supporting faculty, was using 4 machine shops, 2 electronics shops, and dozens of computers in several labs. The organization, managerial and communication aspects of this project soon became nontrivial. To attempt to control some of this chaos the design team created several lead positions and formed itself into subgroups. The subgroups were chosen to be:

Chassis group: which dealt with all aspects of leg, joint, and chassis design, material selection and machining.

Electronics group: which dealt with all electrical systems, including microprocessor construction, motors, motor drivers, sensor design and construction, as well as power supply to the robot.

Controls group: which initially dealt with specifying sensor choices, sensor placement, specifying processor requirements, and writing all software and control algorithms.

The positions on the team were Project Lead Engineer,Control System Group Leader, Electronic System Group Leader, Chassis system Group leader, Librarian, Account Manager, and the Graduate Assistant. The responsibilities of the Lead Engineer were to coordinate the design process between the groups, aid in the design where difficulties arise, and attempt to promote the projects progress. The group leaders were responsible for helping their group members with the design and construction of their prototype sections, requesting needed supplies, equipment, and other support. The Librarians responsibilities were to file all papers, notes, journals, catalogs, progress reports and design documents. The Graduate Assistant was given the responsibility of interfacing between the students and the faculty, handling the teams budget, and purchases, and aiding in acquiring needed equipment and supplies. The Account Manager was charged with aiding new computer users while they set up their accounts and answering computer related questions.

During second semester the positions were filled as follows:

- Lead Engineer—Jon Freise (CIS)
- Chassis Leader—Paul Roesner(ME)
- Electronics Leader—Scott Shute(EE)
- Controls Leader—Tom Hampton(CIS)
- Librarian—Terri Detter(CIS)
- Account Manager—Eric Armstrong(CIS)
- Graduate Assistant—Travis Rhodes(ME)

Part II

Mechanical Design

4 Overview

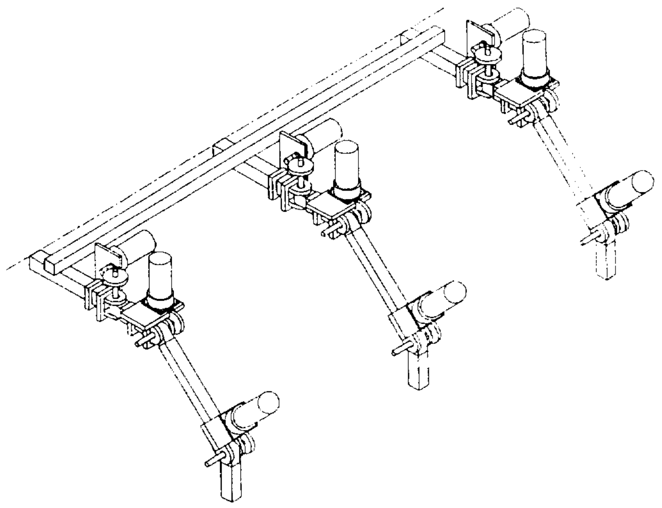


Figure 4.1: Robot Overview

The chassis design team was responsible for the design and construction of the robot's chassis and legs to achieve the long and short term goals of the project. These goals, briefly, were to build a robot that could cross diverse terrain and climb stairs. The goal of the ability to climb stairs caused consideration of designs that would allow the legs to reach high enough and far enough away from the robot to place its legs on consecutive stairs and walk up them. An insect type design, with six legs mounted on a central chassis section, was chosen due to its high clearance, diverse range of motion in its joints, and light weight.

The robot that was built consists of a chassis made from aluminum tubing, to which six legs were mounted. The chassis consists of two long center rails and three cross members bolted to the center rails. One leg is mounted to each end of the cross members. Each leg consists of three joints separated by lengths of aluminum tubing. The first joint (Alpha) is mounted to the end of each cross member. This joint swings in the horizontal direction. The second joint on the leg is the Beta 1 joint. It moves only in the vertical direction. The last joint, the Beta 2 joint, moves in the vertical direction as well. At each joint, an electric motor is mounted, which drives the joints through a worm-worm gear transmission. Figure 4.1 shows the placement of the legs along one side of the chassis, and the placement of the gears and motors at the joints.

5 Leg Design

5.1 Requirements

In the design of the legs of the robot, certain requirements had to be considered. First, the leg had to allow the robot to move at the original design speed of one foot per second. This speed was set arbitrarily to insure that the robot would have a fast response time in real life applications. Next, the legs had to allow the robot to maneuver around or over obstacles in its path. The design also had to provide the robot with the ability to climb a set of stairs.

5.2 Design Considerations

Certain design parameters had then to be considered that would satisfy these requirements. These parameters were the types of motion the

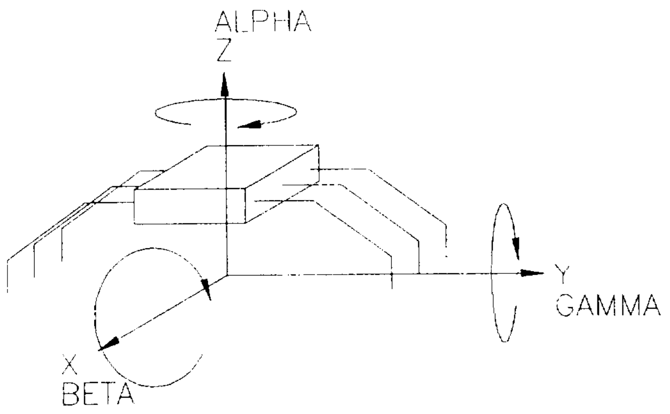


Figure 5.1: Base Coordinate System

legs would use, and the geometry of the legs.

The types of motion that were considered for the legs were linear actuation and rotational actuation. Linear actuation would consist of leg sections that would slide inside one another to extend and contract the leg. Rotational actuation would consist of leg sections that would be attached through joints which would swing through an arc to provide leg movement. Of the two types of motion, rotational actuation was chosen because it provided a less complicated leg design.

After the type of motion was chosen, a base coordinate system for the robot was defined and can be seen in Figure 5.1. With the robot standing up-right, the origin of this coordinate system was chosen to pass through the center of the robot. The vertical axis of the coordinate system is defined as the Z-axis. Rotation about this axis is defined as Alpha rotation. Right to left, or from side to side of the robot is the Y-axis. Rotation about the Y-axis is defined as Gamma rotation. Lengthwise along the robot is the X-axis. Rotation about the X-axis is defined as Beta rotation.

It was then necessary to consider some aspects of leg geometry including: the number of degrees of freedom (either two or three), the number of leg joints (from one to three), and the types of rotation permitted by each joint (Alpha, Beta or

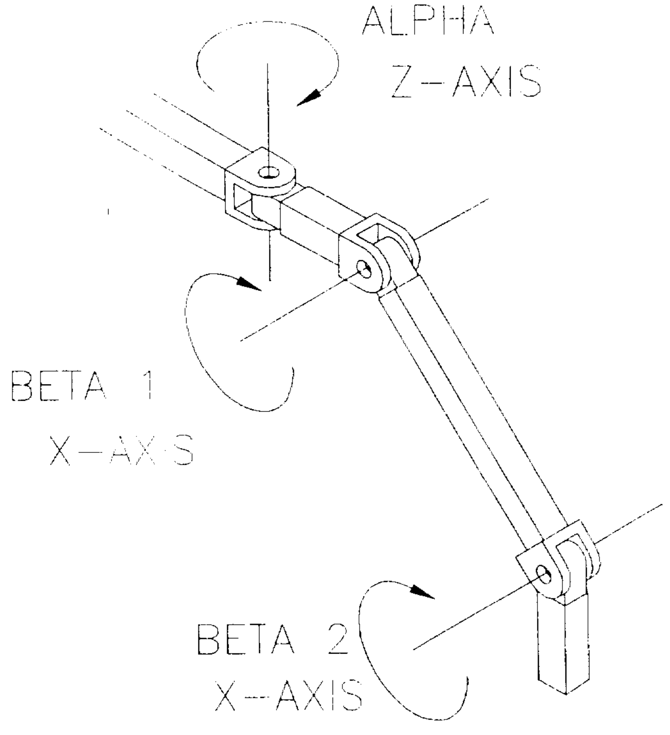


Figure 5.2: Initial Leg Design

Gamma). Joints with multiple types of rotation about a single joint were considered, but ruled out due to confined space requirements and the additional complexity of the joint.

5.3 Initial Design

Of the thirty designs considered, the initial leg configuration shown in Figure 5.2 was chosen. It has three degrees of freedom and a separate joint for each degree of freedom. One of these joints provides Alpha rotation, and while the other two joints provide Beta rotation.

The first joint, the Alpha joint, rotates about the Alpha axis and is responsible for the forward and backward motion of the robot. This means that the speed of rotation of the Alpha joint directly governs the speed that the robot will be able to move, since it is the only joint that moves in the horizontal axis.

The second joint, Beta 1, rotates about the

Beta axis. The Beta 1 joint is responsible for lifting the leg and for raising and lowering the body.

The Beta 2 joint, which is the last joint, also rotates about the Beta axis. Attached to this joint is the last leg section, or foot of the robot. The Beta 2 joint is responsible for stable and controlled maneuvering of the robot. This joint rotates to keep the foot of the leg moving along a straight line with respect to the chassis as the Alpha joint rotates through an arc. This prevents the foot of each leg from having to slide relative to the chassis as the robot moves forward or backward.

This initial leg design was chosen because it provides high clearance capabilities, a large range of motion, and is a flexible design, allowing the use of variable leg section lengths between the joints.

One advantage of this design is that using two joints with Beta rotation allows lateral or sideways motion. This could be useful in tight corners where turning could be difficult or impossible. Another useful feature of this design is that if one of the Beta joints becomes inoperative, the robot would still be able perform many of its functions with the remaining Beta joint. The use of three degrees of freedom also gives the robot the ability to probe with any of the legs. This would allow the robot to sense its surroundings using sensory devices attached to the legs.

The major disadvantage of this design is the lack of Gamma axis rotation. Gamma axis rotation would be useful in situations where the robot had flipped over and needed to maneuver on its back or attempt to right itself by flipping back over. However, the use of Gamma rotation would have led to a much more complicated design, and it was determined that it would not be necessary for our purposes.

With a very basic leg design completed, the leg joint spacings were then determined. This factor would need to be maximized to increase the amount of clearance given to the robot, as well as the speed at which the robot would travel. The joint spacing would also have to be minimized to decrease the amount of torque that would have to be supplied to the joints.

To determine the necessary leg section lengths, several robot models with different joint spacings were constructed using PVC pipe. Trials were then performed, manipulating the models by hand up a set of stairs to determine which joint spacings worked best.

The distance between the Alpha and Beta 1 joints was initially set at four inches. This distance was set at the minimum amount necessary for placement of the power transmission. The distance between the Beta 1 and the Beta 2 joints was arbitrarily set at 12 inches. This allowed about a 12 inch step as the robot moved forward. Through the trials with the models, it was determined that the last leg section should be as short as possible to allow the robot to climb the stairs. However, the longer this leg section is, the less the Beta 2 joint would have to rotate to prevent the foot from sliding. After much discussion, it was decided that this distance should be roughly one-third the distance of the middle leg section. This would provide a reasonable compromise between the amount of clearance height and the amount of rotation necessary at the joint. Therefore, the length of this section was set at four inches.

These trials also allowed the determination of the angles that the joints would need to swing through. It was determined that the Alpha joint would need to swing through an angle of $\pm 30°$ from perpendicular with the chassis. The Beta 1 joint would need to swing through an angle of $\pm 60°$ from parallel with the Alpha leg section. It was also determined that the Beta 2 joint would need to swing through a maximum angle of $70°$ downward from parallel with the Beta 1 leg section, when climbing stairs.

5.4 Final Design

With the basics of the initial design complete, the final leg design was begun. This involved further design of the joints and leg sections including determining actual dimensions and materials.

5.4.1 Joint Criterion

The main goal in the design of the joint was to have a strong, yet light weight, connection that moves freely and smoothly and doesn't require a lot of space. The joint would also require as wide of a range of motion as possible. Because of the complexity of the joint, it would have to be machined, requiring the choice of a material that could be easily machined. An inexpensive material was also desired.

5.4.2 Joint Choice

Two materials that were considered for the joints were steel and 6061 T6 aluminum. Steel is stronger and cheaper than aluminum, but it weighs much more and is more difficult to machine. The 6061 T6 grade of aluminum offered an ultimate strength compatible with the joint design, while retaining good machining characteristics and a light weight. Therefore, aluminum was chosen as the joint material.

A rod and yoke combination, seen in Figure 5.3, was chosen for the joints of the robot because it is a very strong but simple design. The rod and yoke are connected by a steel shaft that was fixed to the rod with a set screw. The rod is designed to fit tightly in the yoke so there is little movement outside the plane of rotation. Roller bearings in the yoke provide smooth rotation of the joint. The ends of the rod and yoke are designed to fit tightly into the leg sections and are fixed with four screws. Thin plastic spacers are also placed between the rod and yoke to prevent wear of these parts caused by rubbing.

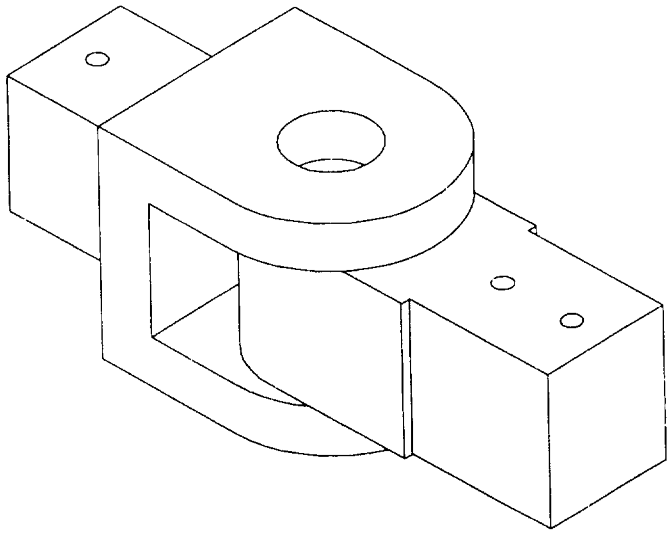


Figure 5.3: Leg Joint

One interesting aspect of the joint design is that the yoke and rod are designed to be interchangeable. For instance, the spacing of the screw holes in the ends of the rod are the same as for the yoke. The distance from the center of the shaft hole in the rod to the end of the rod is also the same as for the yoke. This would allow these joint sections to be swapped without changing the distances between the joints of each leg section. This feature may be useful if different motor mounts and drives are used in future designs.

5.4.3 Stress Analysis

A stress analysis was done based on the final joint design for 6061 T6 aluminum. The equations and calculations for the analysis can be found in the appendix. These equations were solved to determine the maximum allowable forces before failure of the part. These calculations showed that the joints would definitely be strong enough to support the weight of the robot.

5.4.4 Problems

The major problem with joint design was the use of set screws to secure the joint shaft to the rod. In actual operation of the robot, these set screws would loosen, and allow the shaft to rotate within the rod. Using a liquid thread locking compound on the set screws may solve this problem. This might prevent the set screws from backing out of the hole. Another alternative is to drill through the rod and shaft and pin the two together. This would solve the problem, but would make disassembly of the robot very difficult.

5.4.5 Leg Sections

The material for the leg sections between the joints needed to be strong in resistance to both torsion and bending due to the weight of the robot. Materials considered for the leg sections were steel, aluminum, and plastic tubing. One inch square 6063 T6 aluminum tubing with one-sixteenth inch thick walls was chosen because it weighed much less than steel or plastic tubing of the same strength, without costing much more.

The initial distance between the Alpha and Beta 1 joints was four inches to allow room for the motor placement. This length had to be increased by half an inch in the final design to ease assembly and disassembly of the leg section. Since the rods and yokes extend an inch on each side of the tubing, only two and a half inches of aluminum tubing were needed between the joints. The distance separating the Beta 1 and Beta 2 joints was kept at the original 12 inches, so 10 inches of tubing was required. The final distance from the Beta 2 joint to the foot was set at four inches, which required three inches of tubing.

6 Power Transmission

With the final leg design determined, details regarding transmission of power to the joints were addressed. Electric motors had been chosen to supply this power, requiring the design of motor mounts as well.

6.1 Gears

Proper power transmission from the electric motors to the leg joints was necessary to provide a reasonable compromise between the speed of rotation of each leg section and the torque required to rotate each joint under the weight of the robot. For this design, a large increase in the torque supplied by the motor was required to support this weight. Calculations using the weight of the robot to determine the required torque at the joints were done, and are given in the appendix. These calculations were made at the Beta 2 joint which is required to give the most support. Earlier estimates concluded that a required reduction of about 20:1 was needed, but after these calculations were done, the needed reduction was set at 30:1. This would reduce the speed of the robot somewhat, but was deemed necessary for the robot to support itself.

The types of power transmissions that were considered include worm gearing, planetary gear sets using spur gears, and chain drives. For this robot a power transmission consisting of a worm-worm gear combination was chosen. It provides a 30:1 reduction in speed of the motor and a corresponding increase in torque supplied to the leg. In this design, a single threaded worm is pinned to the motor shaft. This drives a 30 tooth worm gear that is pinned to the shaft through which the yoke and rod sections of the leg joints are connected. This shaft is free to rotate in the bearings of the yoke section, but is secured with a set screw to the rod section of the joint. Thus, rotation of the worm gear causes rotation of the rod-end leg section in all the robot's joints, and reduces the rotational speed of the leg joint to about 6 revolutions per minute.

Initial design of the power transmission for the Beta 2 joint differed from the other two joints in that the motor and worm gearing was placed further away from the joint. This design also included a chain and sprocket combination running off the worm gear shaft and driving another shaft at the joint. This design was considered because it would locate the mass of the Beta 2 motor closer to center of the chassis, increasing the stability of the robot with its legs extended. This design was not adopted for the final design, however, because it would have added considerably to the weight and complexity of the design.

The use of worm gearing was chosen because it meets all the original design criterion. The design is very simple, consisting of only two gears, one extra shaft and two bearings. A similar transmission consisting of spur gears or chain drives would be too large, and would need several additional shafts and bearings to achieve the necessary reduction. An additional benefit of using worm gearing is that the joints are self-locking and will support the weight of the robot without any power to the motors. This feature would not be available with the use of spur gears or chain drives without additional locking mechanisms.

Various catalogs from Stock Drive Products, Berg, and Chicago Gear Works were used to make the gear selection. The worm chosen was from Chicago Gear Works. It was the only one available that was close to fitting the 8mm motor shaft. A worm gear of the same pitch and pressure angle was then chosen to meet the required gear reduction. This worm gear had a five-sixteenths inch center bore, requiring machining to fit the three-eighthes inch gear shaft. The gear reduction of a worm-worm gear combination is computed by dividing the number of teeth on the worm gear by the number of threads on the worm. Therefore, thirty teeth on the worm gear gave the required 30:1 reduction for the single threaded worm.

6.2 Motor Mounts

The motor mounts for the robot were designed to hold the motor firmly to the leg joints and to allow the worm on the motor shaft to mesh properly with the worm gear. It was also designed so that it would not interfere with each joint's rotation. Another design criterion was to use the same design for all the motor mounts to ease the manufacturing process. The motor mounts were also designed to allow the addition of slightly larger worm gears so that a higher gear reduction may be used if needed.

The motor mount design used in the robot can be seen in Figure 6.1. It consists of a flat aluminum mount plate to which the motor is secured. The mount plate is held tightly to the joints by two U-shaped clamps. The larger clamp is secured with screws to hold the mount plate to the yoke section of the leg joint. A spacer bar is used with screws in the smaller clamp to hold the mount plate against the tubing of the leg section. A small amount of clearance between the clamps and the mount plate exists, so that the motor mount may be tightened completely to the joint. Since the mount is only clamped to the leg, it may be slid back along the leg to accommodate a larger worm gear.

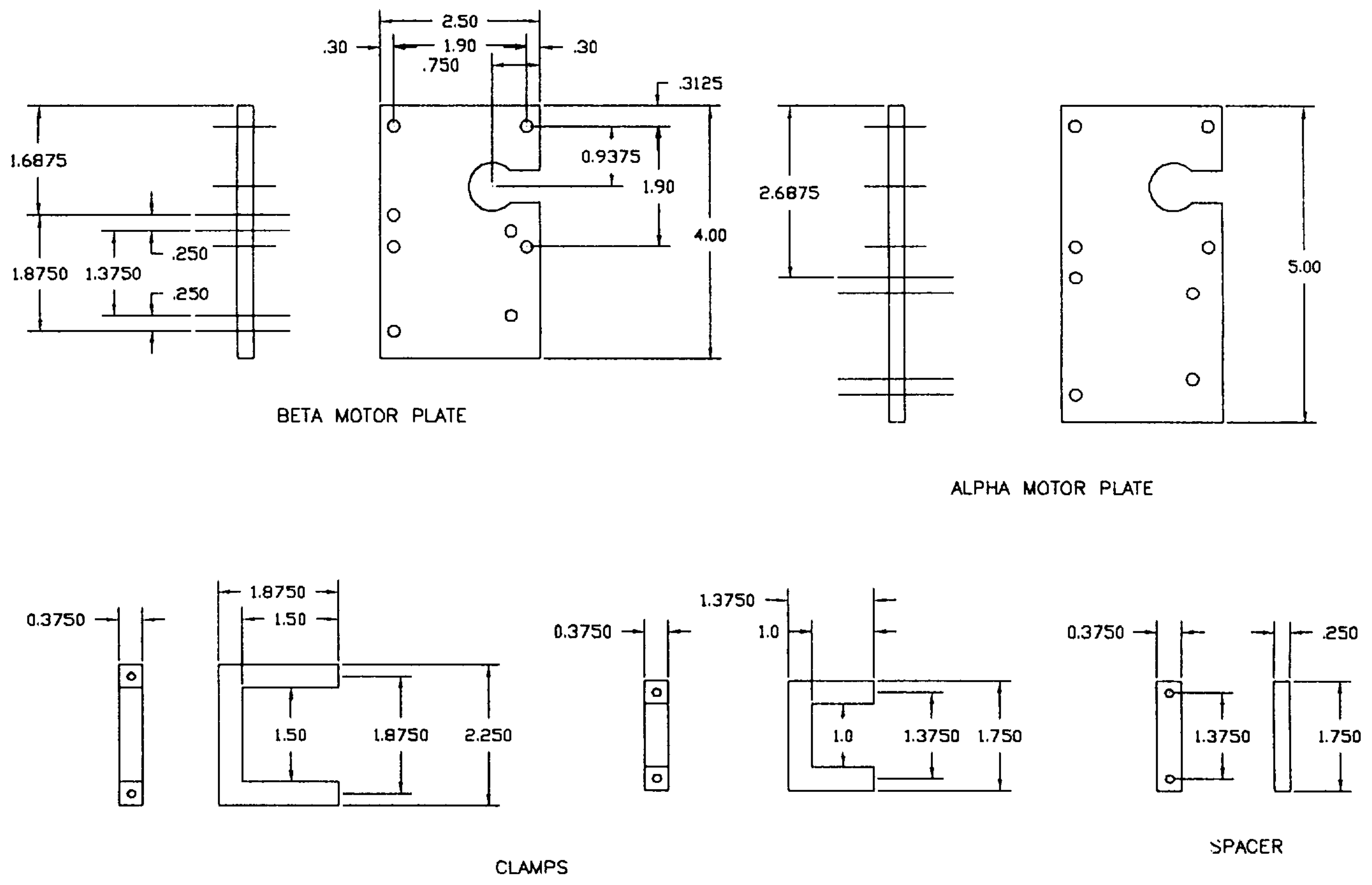


Figure 6.1: Motor Mounts

The design of the motor mount limits the rotation of the leg sections to a maximum of ±80°. The Alpha motor mount design was changed somewhat from the other mounts to permit the Alpha joint to swing through its the full range of ±80°. This feature was not one of the original design criteria, but was added to increase the robot's maneuverability in stair climbing. To do this, the Alpha mount plate was lengthened by one inch to raise the Alpha motor further from the joint. This was necessary to avoid the collision of the Beta 2 mount plate with this motor as the Alpha joint rotates through this larger angle. This design change required the use of longer worm gear shafts at the Alpha joints as well.

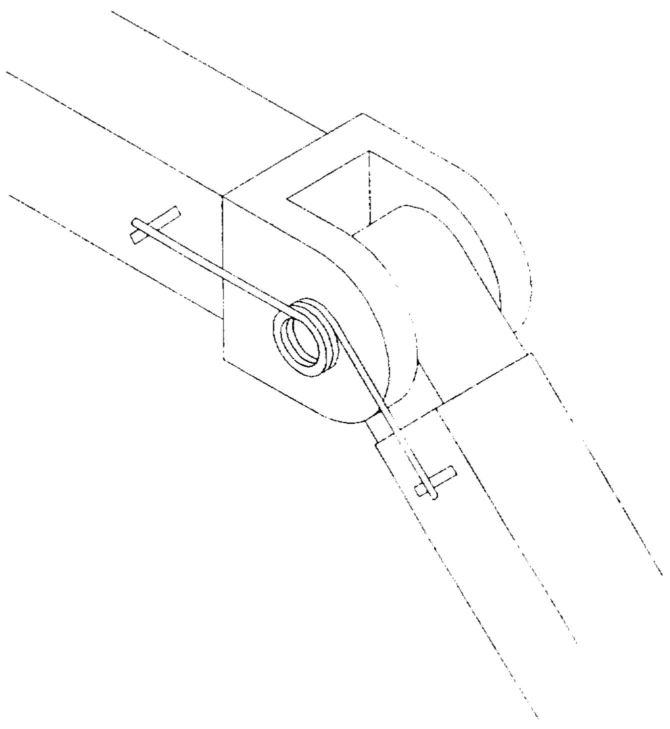


Figure 6.2: Counter Spring

6.3 Problems

A major drawback in using the worm-worm gear transmission is the low efficiency, or ratio of power transmitted through the gear set versus the amount of power supplied to the motor, that is associated with this type of gearing. This factor may be caused by the friction of the worm sliding across the teeth of the worm gear, or by a slight misalignment of the worm and worm gear. Through tests during the assembly of the robot, efficiencies as low as 25 percent were measured. Although the torque supplied to the joint is limited, it is still enough to lift the weight of the robot.

Solutions to this problem may involve proper lubrication of the gears, or a redesign of the motor mount to provide exact alignment of the worm and worm gear. To reduce the amount of torque required at the joints, counter-springs could be added at the joints as seen in Figure 6.2. Since the weight of the robot acts downward, the most torque is required of the motors when lifting the robot up. Counter-springs could be loaded to provide a couple equal or nearly equal to that produced by the weight of the robot. This would require the motors to produce more torque when raising the leg, but this torque could be balanced so that the overall torque would be less than at present. The use of counter- springs was not included in the final leg design because it was calculated that the motors would always have enough torque to lift the robot. Still, this feature could be easily added to the existing design.

Another problem with the power transmission is the excessive play of the motor shafts in and out of the motor housing. This movement allows the joints to rock back and forth under the weight of the robot. Solutions to this problem may involve a redesign of the motor mounts to incorporate thrust bearings on the motor shaft.

7 Chassis

The robot chassis needed to be as rigid as possible for smooth operation and consistent, accurate feedback of information on component locations. Resistance to torsion is necessary due to the various walking gaits required for different terrain. The chassis also needed to provide a stable platform for mounting the electronic hardware and legs to the chassis. The chassis needed to be long enough to allow for the horizontal movement of the legs, without the legs hitting each other, and short enough to fit up a set of stairs. The chassis had to be narrow enough to fit up a set of stairs also, but wide enough to allow the electronic hardware to be mounted.

One initial design considered was to use aluminum honeycomb composite plates. These plates were in an enclosed box configuration to achieve torsional rigidity, while conserving weight. These plates would resist bending moments as well as compression and tension forces. Tensioning cables would also be used between selected joints to keep the box from collapsing. The Alpha joints would be bolted to brackets extending from the plates. Although this design had the advantage of being lightweight, it was not chosen because it was expensive, and difficult to build.

Another chassis design was considered using one inch square 6063 T6 aluminum tubing with one-sixteenth inch thick walls. Two forty inch center rails would be placed along the length of the robot. They would be separated by a width of four inches. Three fourteen inch cross members would then be bolted to the to the center rails. The yoke sections of the Alpha 1 joints would fit into the tubing of the cross members. This system had the advantage of being easily altered by using different lengths of center rails or cross members. It could also be built simply and inexpensively, and still be lightweight. The closed shape of the tubing would make this design extremely resistive to torsion.

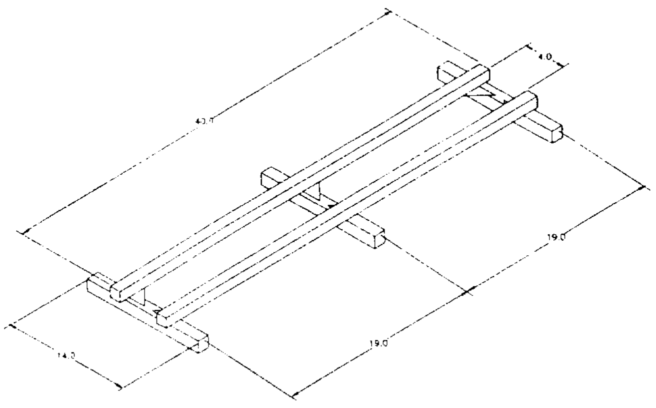


Figure 7.1: Robot Chassis

This chassis design was selected and is shown in Figure 7.1. The length of the center rails was determined by the needed distances between the legs in the arc of motion required for the desired gaits. This length was found to be forty inches. The separation width of the center rails was chosen to be four inches because it gave adequate room for the electronic hardware to be mounted. The cross members were chosen to be fourteen inches long. This length allowed the robot to fit up the stairs, while allowing room for the electronic hardware.

One inch square 6063 T6 aluminum structural tubing was chosen for the center-section because it would not only give excellent torsion resistance, but its flat faced contour allowed simple connections and integrations of other components. The aluminum insured low weight with adequate strength, as total weight of the robot was a major design parameter.

The chassis was designed to be bolted together due to the anticipation that the design could change over the course of the project, thus making adaption simpler. Gussets were added in the interior angles of the frame to stop any relative shifting between the chassis tubing. These gussets were made of one-sixteenth inch thick steel plate.

7.1 Problems

A problem that needs addressing is that of chassis rigidity. The present bolted design has advantages in that it was easily adapted and disassembled. Although steel gussets were added to stiffen the chassis, relative shifting of the bolted parts still occurs. The main chassis frame needs to be of welded construction. This would eliminate lack of rigidity in the frame due to bolting.

7.2 Electronic Mounts

The electronic parts were mounted to the frame by first connecting all components to Plexiglas and bolting the plate to the frame. This allowed the connection of all the needed hardware to the frame with a minimum number of holes.

The electronic equipment is somewhat fragile and will need to be protected in the case of an accident. Fiberglass or carbon fiber protection pieces could be fabricated to provide this protection. The composite pieces could be very lightweight, adding little to the overall weight of the robot. They would be fabricated in the Composites Materials Lab, room 25 in Durland Hall, under the supervision of Dr. Hugh Walker.

8 Machining

The parts on the robot that were machined were, the joints, gears, motor mounts, and leg sections. The machining took the majority of the time that was spent on the construction of the robot. Speed and accuracy in machining each part was the main concern. Some of the machining was done in the Automated Machines Lab located in the basement of Durland Hall. This lab had a programmable CNC milling machine that would enable quick and more accurate production of the parts. The equipment is owned and operated by the Advanced Manufacturing Institute (AMI) at Kansas State University. AMI allotted twelve hours of programming and machining time in the Automated Machines Lab. For this reason only the joint sections could be machined there. Labs used for machining the other parts were the Mechanical Engineering Shop in room 23 in Durland Hall, and the Production Processes Lab in room 21 of Durland Hall.

8.1 Joints

The yoke and rod sections were first cut from one and a half and one inch square 6061 T6 aluminum bar stock, to their respective lengths using a power hacksaw in the Mechanical Engineering Shop. Due to the complexity of the parts, the yoke and rod sections of the joints were then machined on the CNC milling machine in the Automated Machines Lab. To begin this process, the Autocad drawings for the yoke and rod were exported as DXF files.

This allowed the AMI machine operator, Dan MacAnerney, to bring the drawings into another program that would create tooling paths and holes that would be drilled on the various views. The information created in the program was downloaded to the four axis CNC milling machine. During this process the machine created the code necessary to carry out the tooling and drilling. For the rod and the yoke there were three different setups. Each piece needed to be placed in the machine three different ways in order to contour, face, and drill all the required features.

Since our allotted CNC milling time of twelve hours for the Automated Machines Lab had been used up, the rest of the machining for the joints, and for the other parts, was done by hand and by using the lathes, milling machines, and drill presses in the Production Processes Lab, and the Mechanical Engineering Shop.

The screw holes in the ends of the rods and yokes were tapped on both sides using a 6-32 hand taps. Seven-thirtysecond inch holes were then drilled into the shaft ends of the rods, and tapped using 1/4-20 hand taps.

8.2 Gears

The worms and worm gears also required machining, since gears with the proper bore dimensions could not be found. The worms had to be reamed to allow them to fit the 8 mm diameter motor shafts. This was done on a lathe in the Production Processes Lab using an O-sized reamer. The worm gears were also drilled and reamed to a three-eighth inch bore on the lathe.

8.3 Motor Mount Plates

The next machining project was the motor mount plates. The quarter inch thick plates were cut from six inch wide 6061 T6 aluminum stock with a hydraulic shearing machine in the Production Processes Lab. A template for the Alpha mount plate was made using one of the aluminum plates. Use of the template insured that all mount plates would be the same. Holes for the motor were marked on the template using a transfer punch and then drilled. Holes for the mount clamps were also marked and drilled on the template. Using the template for a guide, the holes were then drilled in the six Alpha mount plates, by clamping two plates at a time to the template in a vice. Holes for the Beta 1 and 2 mount plates were then drilled in the template. This was done by drilling the mount clamp holes three-fourths of an inch up from the others. Holes for the twelve Beta 1 and 2 mount plates were then drilled using the template as before. The holes in all the plates for the clamps were then counter sunk on one side.

8.4 Motor Mount Clamps

The clamps were made from square 6061 T6 aluminum bar stock, using a two and a half inch square block for the large clamp, and a two inch square block for the small clamp. They were first milled to the proper outside dimensions on a face cutter. Channels were then milled out of the middle of the blocks. Holes were spaced and drilled for the screws that would mount the clamps to the mount plate. The blocks were then cut into three-eighth inch wide pieces to make the individual clamps. The holes were tapped into the clamps using 6-32 hand taps. Two inch long, three-eighth inch wide spacer bars were cut from left over one-fourth inch aluminum plate on a band saw. Holes corresponding to the holes in the small clamp were then drilled in the spacer bars.

Quantity	Length (in)	Part
2	40	center rails
3	14	cross members
6	2.5	Alpha leg sections
6	10	Beta 1 leg sections
6	3	Beta 2 leg sections

Table 8.1: Square Tubing lengths

8.5 Tubing Sections

The aluminum tubing was cut to the lengths given in table 8.1.

Holes for joint screws were then drilled at the end of the cross members and leg sections. This was done using a template made from a scrap piece of steel tubing to insure proper spacing of the screw holes. Holes corresponding to the screw holes in the yokes and rods were drilled in both sides of the template. By sliding this template over the ends of the tubing, these holes were then drilled in the tubing. Holes were also drilled in the cross members and center rails to allow them to be bolted together.

Gussets for the chassis were cut from a one-sixteenth inch thick steel sheet. Holes were then drilled in the gussets to allow them to be bolted between the cross members and center rails.

8.6 Problems

There were a few problems encountered during the machining of the parts. The machining process was slow and sometimes cumbersome. The milling machine in the automated machining lab had problems with the offsets for the coordinated axis origin. Several different setups had to be experimented with to try and find a suitable position for the part in the machine. The rest of the machining was done by manual operation of the milling machines in other labs which was much more time consuming. It also caused small variances is size for some of the parts, which does not appear to be a serious problem.

Another machining problem concerned the hand tapping of the screw holes for the yokes and rods. This was a slow and cumbersome task, requiring several minutes to complete each part. The 6-32 taps were very fragile, and it was easy to break the taps by twisting too hard. This would also require removal of the broken tap from the hole which was not easy to do. Self-tapping screws would have eliminated this problem, and were considered. However, these screws would strip out the threads in the aluminum if removed often. Since it was likely that the robot would have to be disassembled occasionally, they were not used.

9 Robot Assembly

The assembly of the robot was completed in the Mechanical Engineering Shop in room 23 of Durland Hall. The assembly consisted of several smaller assemblies including: press-fitting the bearings in the yokes, mounting the worm gears to the shaft, mounting the worms to the motor shafts, assembly of the chassis, assembly of the motors to the mounts, assembly of the leg sections, assembly of the legs to the chassis, and mounting of the control relays.

9.1 Press-fitting the Bearings

The needle roller bearings were pressed into the holes of the yoke sections by first tapping them lightly into the holes with a plastic mallet. The yokes were then put in a vice with a scrap piece of aluminum placed between the flange of the yokes so that they would not bend as the vice was tightened. The vice was then tightened, pressing the bearings into the holes.

9.2 Mounting Worm Gears

The first step in this procedure was to secure locking collars to the ends of each of shafts. These locking collars were used only as spacers and were removed at the end of the procedure. The worm gears were slid onto the shafts up against the locking collars, with the hubs of the worm gears on the opposite side of the locking collars. The gear-shaft assemblies were then placed one at a time in a special drilling jig made from a scrap piece of aluminum. The jig had a one-eighth inch pilot hole on top and a narrow slit on the bottom. The assemblies were placed in the jig with the shaft running the length of the jig, and the hub of the gear fitting inside it. The jig was then placed in a vice with the one-eighth inch pilot hole facing upwards. Tightening the vice allowed the jig to clamp around the gear-shaft assembly. One-eighth inch holes were then drilled completely through the gear and shaft by guiding the drill bit down the pilot hole. The vice was then loosened and the assembly removed from the jig. A one-eighth inch reamer was then used to clean out the hole, and a one-eighth inch by three-fourths inch long solid steel dowel pin tapped into the hole with a plastic mallet.

9.3 Mounting the Worms

The first step in this procedure was to place the worms on the motor shafts with the hubs of the worms one-fourth of an inch from the motor base. The holes in the worm hubs were then marked on the flat side of the motor shafts with a scribe. The worms were then removed and the scribe marks center-punched. The worms were replaced on the shafts and the motors placed in a small vice one at a time. A vee block was used to support the worms on the shafts. One-eighth inch holes were then drilled through the worms and shafts. A one-eighth inch reamer was used to clean out the holes, and one-eighth inch by one-half inch long solid steel dowel pins tapped into the holes with a plastic mallet.

9.4 Chassis

The three cross members of the chassis, which support the six legs of the robot, were bolted to the two longer center rails, with the steel gusset members bolted between them. Six of the yoke sections were then placed in the ends of the cross members. The yokes were then fastened into place using the three-eighth inch long 6-32 slotted pan head screws.

9.5 Alpha Mounts

This procedure consisted of first bolting the mount plates to the yokes on the cross members with the three-fourths inch long 6-32 countersunk Phillips head screws, and with the spacer plate inserted between the small mount clamps and the tubing. The six Alpha motors were then mounted to the mount plates by first sliding the motor shaft through the slots on the mount plates and then securing the motors loosely using the 4 mm by 12 mm long countersunk Phillips head screws.

9.6 Beta Mounts

The assembly of these mounts began by sliding the motor shafts through the slot on the mount plates. Then the top screws for the mount clamps were added, with the spacer plates inserted between the small clamps and the mount plates. These top screws were tightened fully, then backed off one-quarter turn to provide the proper clearance between the mounts and the tubing. The motors were then secured to the mount plates with 4 mm by 12 mm long countersunk Phillips head screws. Then, the lower screws on the clamps were added and loosely tightened.

9.7 Leg Sections

Yoke sections were added to the ends of the tubing for the Alpha and Beta 1 legs sections and secured with the three-eighth inch long 6-32 slotted pan head screws. The Beta 1 motors and mounts were then slid onto the Alpha leg sections, and the Beta 2 motors and mounts slid onto the Beta 1 leg sections. Then, the rod joint sections were fastened to the ends of the Alpha, Beta 1, and Beta 2 leg sections using the same 6-32 pan head screws.

9.8 Legs

The legs were assembled starting at the Alpha joints. Plastic spacers .010 inches thick were placed on both sides of the rods as they were inserted into the yoke sections on the cross members of chassis center-section. Locking collars, and .040 inch thick plastic spacers were slid on to the worm gear shafts and set loosely against the hubs.

The worm gear shafts were then slid through the bearings in the yokes and through the holes in the rods. The motor mounts were slid up against the shafts and the two gears were engaged. The locking collars were then tightened against the top of the yokes. Set screws (1/4-20 by one-half inch long) were inserted in the rod and tightened, leaving an impression on the shafts. These set screws were then removed, the motor mounts slid back, and the gear shafts taken out.

The impression on the shafts left by the set screws were center- punched and then drilled to one-eighth inch into the shafts with a one-fourth inch bit. The shafts were then reinserted into the joints. The impressions in the shafts were sighted through the tapped holes in the rods, and the set screws reinserted and tightened fully. Locking collars were then secured to the shafts on the bottom of the yokes. The motor mounts were slid up against the joint until the gears engaged, and secured by tightening the screws on the clamps. In this manner, the Beta 1 and Beta 2 sections were added to the legs. Rubber chair leg cushions were attached to six three inch sections of three-fourths inch PVC pipe, and the pipes secured inside the Beta 2 leg sections.

9.9 Mounting the Control Relays

The control relays were mounted onto an 8 by 10 inch piece of one- fourth inch thick Plexiglas using one inch long 8-32 screws and taps. The Plexiglas was then mounted to the center-section of the chassis using the same screws and taps.

9.10 Problems

The biggest problem encountered during assembly involved the mounting the Beta 1 and Beta 2 motors. Because of the motor mount design at these joints, only the lower mount clamp screws may be tightened fully to secure the motor to the joint. The heads of the upper clamp screws cannot be reached after the motor is secured to the mount plate. This factor did present a problem of mount slippage, requiring the tedious process of loosening the motor from the plate, tightening the top screws slightly, and re-tightening the motor to the plate. This problem was discussed during design, but considered a minor inconvenience necessary in placement of the motor as close to the joint as possible. Still, this problem should be addressed in future motor mount designs.

10 Critique/Future Designs

The major problems and possible solutions with present design are discussed further in this section. This will lay some groundwork for future designs, having gained invaluable experience in the construction and implementation of ideas in this portion of the project.

The torque losses of the present drive train through the worm-worm gear transmission greatly limit the lifting ability of the robot. The weight of the robot causes a constant downward force on the legs and drive train. Implementation of torsional coil springs at the joints to combat the torque caused by gravitational forces is a possible solution. These springs would be placed on the Beta 1 joint and could be implemented with moderate ease. With experimentation, these springs could possibly be chosen to take a great portion of the stress off the drive train. Ideally, the springs would create a resisting torque close to the torque needed to raise a portion of the body.

There was also a problem with the motor mounts sliding on the tubing, thus allowing separation of the worm gears. This allowed undesired movement in the Beta 1 joint. As mentioned earlier, it may be a result of insufficient tightening of the upper clamp screws on the mounts. This slippage would eventually cause high stresses on the teeth of the gear train, possibly causing failure of these parts. One possible solution is to use hex head screws to tighten the mount clamps. More torque could be applied to the hex head screws, to tighten the clamps without stripping the screw heads, than with the Phillips head screws. Another possible solution is to cut each of the mount clamps in two, and drill and tap each half. Screws could then be used to make the clamps adjustable in both clamping axes. This would result in a better clamping force, and would prevent movement of the mounts.

Part III

Electronic System Design

11 Overview

The electronics group had the responsibility of the design and construction of all electronic and electrical components on the robot. This included everything from the microprocessors, sensors, wiring, manual control panel, to the power supply. However most of the electronics were concentrated in the control of the motors and joints. (see fig 11.1)

11.1 Motion Control

The motion planning and decision making software for the robot runs on three 16MHz MC68HC16Z1 microprocessor boards. These boards were purchased complete from Motorola. The only needed additions to these boards is the adding of the serial network connections discussed in the controls group network section.

Each leg of the robot is controlled with an HC11 micro-controller. The HC11 is a 2MHz 8bit processor with a 16 bit wide address bus. The HC11 has had extensive Input/Output additions to handle all the needed sensors and motors. Many special purpose and latch chips have been mapped into its memory space. This mapping allows the HC11 to read and write to these chips as though they were normal memory locations. The HC11 also has eight Analog to Digital converter channels, with which it can read the leg's analog sensors. (see fig 11.2)

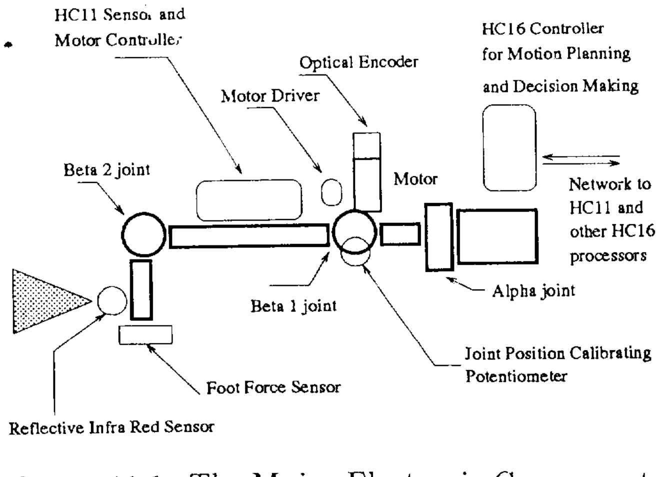

HC11 Sensor and Motor Controller
Optical Encoder
HC16 Controller for Motion Planning and Decision Making
Motor Driver
Beta 2 joint
Motor
Network to HC11 and other HC16 processors
Alpha joint
Beta 1 joint
Joint Position Calibrating Potentiometer
Foot Force Sensor
Reflective Infra Red Sensor

Figure 11.1: The Major Electronic Components

11.1.1 Position Sensing

Because the position of the joints are critical, it is sensed in several ways. First, each joint motor is equipped with a high resolution optical encoder. The optical encoders are read using a special purpose chip, the HCTL-2016. This is the primary way the robot tracks it's joint's positions. Unfortunately, due to the nature of the optical encoder it cannot tell the absolute position of the joint. The encoder is like an odometer, it does not tell you where you are but how far you have moved. The starting point must be known to calculate where the joint has moved to. The starting point is not known when the robot is powered on, so to find the starting point the robot is equipped with a potentiometer mounted to each joint's shaft. As the joint rotates the potentiometer is rotated and varies in resistance (like a volume knob). This resistance value (volume) is measured by the HC11's Analog to Digital converter to determine the joints absolute position. In addition, as an emergency failsafe, each joint is also equipped with two limit switches. If for some reason the motor should try to drive the joint into an extreme position, a limit switch would be set off and it would disable the motor driver from driving the motor further (The motor can still be backed in the safe direction). The HC11 has a input latch for reading the values of the limit switches, so it knows when the joint has been rotated to far.

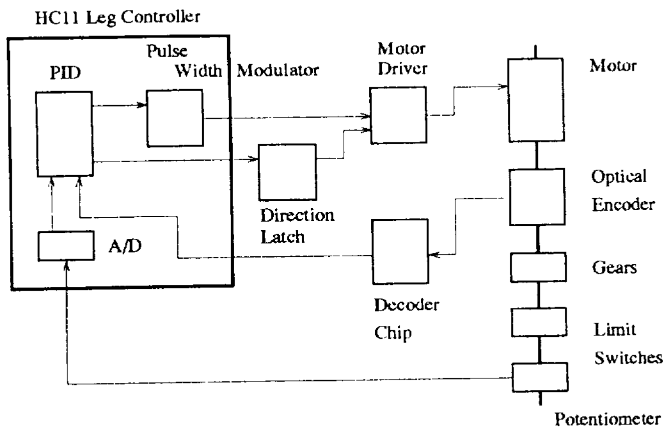


Figure 11.2: HC11 Leg Controller and Extensions

11.1.2 Motor Control lines

The HC11 can control the direction of the motors by writing values into a dedicated output latch. This latch is connected to the 3 high power motor driver chips, and determines which direction current flows through the motors. Three of the HC11's special timer port pins are also tied to the motor drivers. Each pin is used to switch one of the motor drivers (and it's motor) on and off. This feature is used to control the speed of the motors with a technique referred to as Pulse Width Modulation.

11.1.3 Working Together

When the robot powers on, the HC11 controller reads the three joint positions from its potentiometers and calibrates the optical encoders. (see fig 11.2) It also reads the foot force sensor, and the I/R obstacle sensors and sends all this data up to the motion planning software on the HC16's. The motion planning software does its calculations and sends back the positions that it desires the joints to be in. The HC11 has a routine running on it called the Proportional Integral Derivative control routine, which compares the joints current position with the desired position and calculates the direction and speed the motor should move with. The motor direction is programmed directly into the motor direction latch. The motor speed is fed to another routine running on the HC11 called the Pulse Width Modulation routine. The PWM routine uses the special timing pins of the HC11 that are tied to the motor drivers to control the speed of the motors. New sensor information current joint positions are sent up to the motion planning software 25 times per second, and the newly commanded joint positions are sent back just as often.

12 Motors

12.1 Discussion

To propel the robot, a self-contained, lightweight and inexpensive source of power generation is required. DC motors were chosen for this task. For maximum efficiency and power, a motor is required at each joint of the robot, therefore 18 motors are needed. In order to determine the power required of each motor, the highest torque requirement was calculated. This worst-case load is achieved by the Beta 1 motor during the tripod gate, and is calculated to be half the weight of the robot. The required torque in this scenario is based on the following assumptions concerning the component weights (see table 12.1).

To find an expected torque for this situation, the one foot link between Beta 1 and Beta 2 joints was used.

$$Torque = (1ft.)(40.5lbs.)(0.5) = 20.25ft. - lbs.$$

Item	Weight
Aluminum tubing 1" square	6 lbs.
Hinges	6 lbs.
Motors (15 x 1.5 lbs. each)[1]	22.5 lbs.
Bolts, shafts, processors, etc.	6 lbs.
Total estimated weight	40.5 lbs.

Table 12.1: Robot weights

Originally, 3.8 V motors taken from cordless Black and Decker screwdrivers (model SD 2000) were included in the design. These motors were reported by the manufacturer to have enough torque (40 ft.-lbs.) and were inexpensive ($26 for the entire screwdriver unit). A major drawback in using the screwdriver motor was in the mounting of the motor to the robot. Because of its shape, which included the screwdriver's gear reduction system, it was very difficult to design a mount for the motor.

The motor that was chosen for the final design is a 24 V motor produced by Matsushita Electric and distributed by Servo Systems. This motor has several advantages over the 3.8 V screwdriver motor and costs about the same ($30).

The advantages of 24 V motor over 3.8 V screwdriver motor are listed below:

- The voltage can be varied around a much larger range.
- The motor driver chips of 3.8 V are more expensive than those of 24 V ($30 vs. $3.00).
- The screwdriver motor and gear reduction are not one unit.
- A mounting face plate is included on 24 V motor.
- The 24 V motor includes a built-in optical encoder.

[1]In tripod gait, the weight of the three Beta 2 motors would be supported by the Beta 2 link sitting on the ground.

The built-in optical encoder of the 24 V motor is a major advantage, as it will be able to supply digital data relating exact relative positioning. To achieve this using the screwdriver motor would involve the installation of an optical encoder on the shaft of the motor, which would have added to the bulk of the machine and would have incurred an added expense.

The disadvantages of 24 V motor compared with 3.8 V screwdriver motor are listed below:

- The 24 V motor is heavier - 1.5 lbs. compared to 6 oz.
- The 24 V motor is larger - 7" long by 2.375" diameter compared to 3.5" long by 2.25" diameter.

The main consideration in motor choice is the torque it produces relative to its size and cost. Black and Decker claims that the model SD 2000 will produce 40 ft.-lbs. It is believed that this figure may be considered the locked rotor torque. The 24 V motor is rated for 21.8 ft. lbs. at 185 rpm, which will be the operable speed. The locked rotor torque for the 24 V motor is 33.3 ft.-lbs. The 24 V motor has a rated current of 5 A. This means that the motor is capable of safely handling (24V)(5A) = 120 Watts of power. In order to accept the same input power, the screwdriver motor would need to carry 24 A. Considering the physical size of the wire and motor from the screwdriver configuration, the claim of 40 ft.-lbs. appears misleading.

Based on the considerations mentioned above, the 24 V motor was chosen.

12.2 Critique of Motors

The motors ordered from Servo Systems are functional. They have been tested to the extent that one 12 V supply can operate at least three motors. However, the worst-case torque analysis is faulty. The weight of the current robot

is 70 lbs. Also, the amount of loss due to friction in the gear train, which was estimated to be negligible, is 75 percent without lubrication. Assuming a loss of 75 percent, the required torque for worst-case would be 140 ft.-lbs. This is far beyond the capabilities of the motors that are currently employed. Using a lubricant on the gears could possibly reduce the loss to 50 percent, but this is still more torque (70 ft.lbs) than the motors can produce.

The required torque needs to be researched further. This current design is adequate for horizontal walking, but is deficient if expected to climb stairs or rugged inclined terrain.

13 Position Control

13.1 HC11

13.1.1 Requirements

The requirements for the digital I/O included being able to set six single bit output lines. Input was required on six single bit inputs, and three 16 bit devices. These latter three are the Quadrature Decoders, and have a tri-state 8 bit interface. The other requirement was to keep a low parts count.

13.1.2 Implementation

The HC11 board digital I/O capability is memory mapped. The solution that was implemented was given the address space of $9000-$9fff. The single bit inputs and outputs were were grouped together into two 8-bit latches. Thus reducing the need for six separate latches for each input. Address decoding is done by a Quad-input AND and a 3-8 decoder. Figure 13.1 gives a block diagram of the electronics. Figures 13.2, 13.3 and 13.4 show the detailed views of the system. Table 13.1 shows gives the parts list of the parts that are on the HC11EVBU board. Table 13.2 shows the addresses that are currently used. Since Address lines 8,9,10, and 11 are not used in decoding, the addresses overlap on 16 byte boundaries.

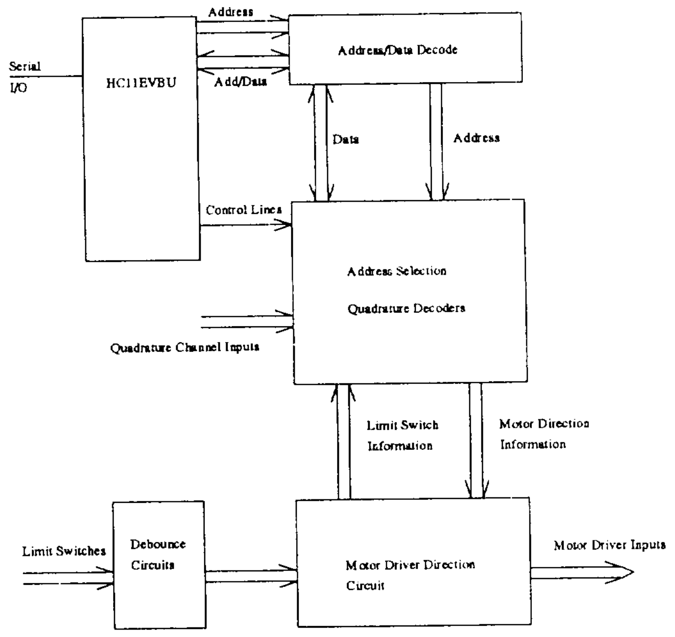


Figure 13.1: Block Diagram of Electronics System

13.1.3 Outputs

The motor directions and reset lines are set by writing to address $9000. The 8-bit byte is split up according to Table 13.3. The outputs once set by a write command will stay set until the address is written to again. The outputs on bits 5, 6, and 7 should normally be high. Setting them low will hold the quadrature decoders in the reset mode until the bit has been set back to the high state.

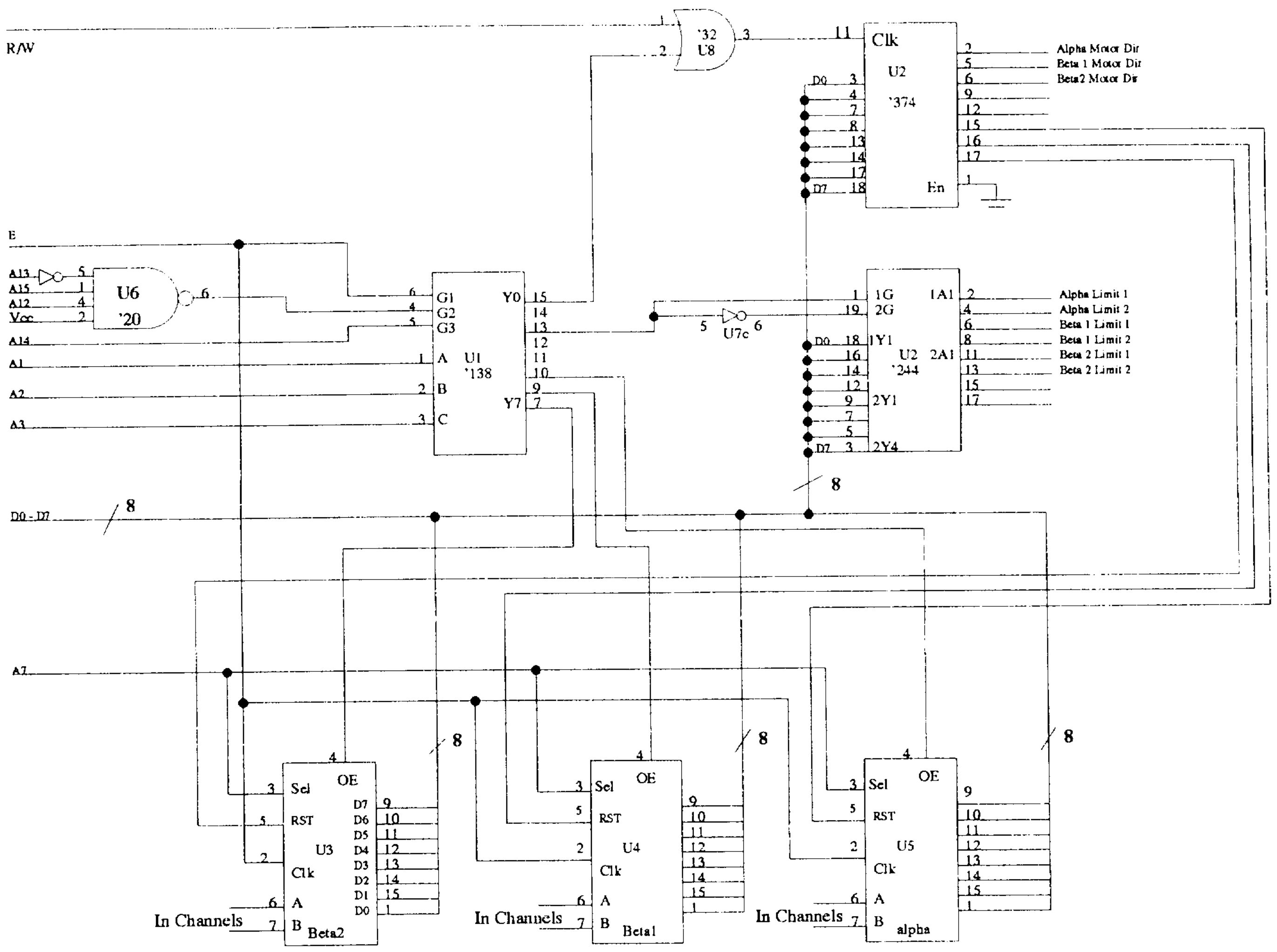


Figure 13.3: Address Decoding and Quadrature Decoder Circuit

Address	Direction	Description
9000	Outputs	Motor Dir. /Quad-decode reset lines
9002	Inputs	Limit Switches
9005	Input	Alpha Quad decode position Low byte
9006	Input	Beta1 Quad decode position Low byte
9007	Input	Beta2 Quad decode position Low byte
900e	Input	Beta1 Quad decode position High byte
900d	Input	Alpha Quad decode position High byte
900f	Input	Beta2 Quad decode position High byte

Table 13.2: HC11 I/O addresses.

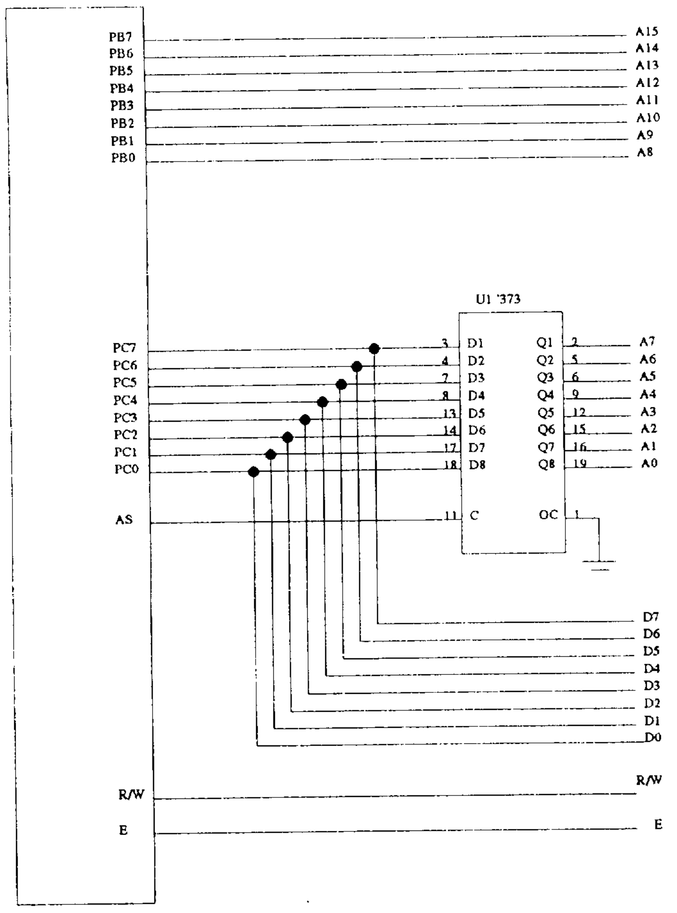


Figure 13.2: Address/Data Decode Circuit

Label	Type	Description
U1	74HC138	3-8 decoder
U2	74HC374	S-R latch
U3 - U5	HCTL-2016	Quadrature Decoder
U6	74HC244	Dual 4-in Tri-state Latch
U7	74HC04	Hex Inverter
U8	74HC32	Quad 2-Input Or
U9	74HC373	Latch
U10	74HC08	Quad 2-Input AND
U12	74HC08	Quad 2-Input AND

Table 13.1: Parts List

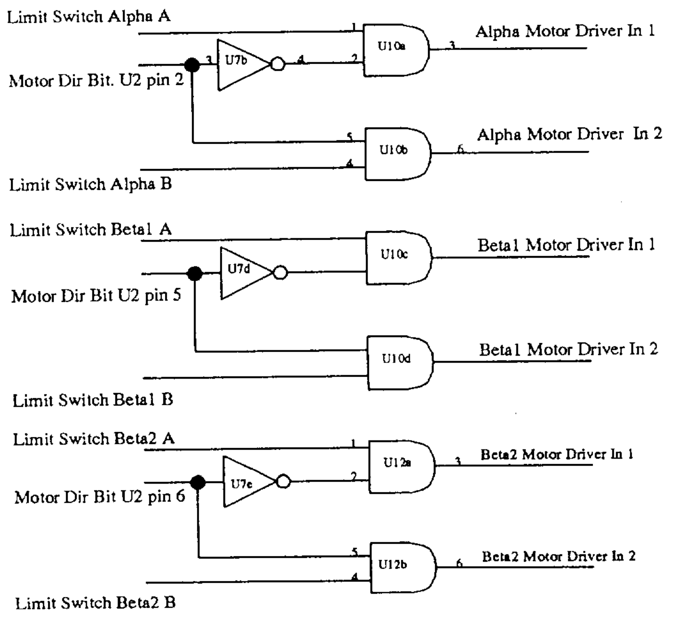


Figure 13.4: Motor Driver Enable Circuit

Bit	Line	High	Low
0	Alpha Motor Dir	CW	CCW
1	Beta1 Motor Dir	CW	CCW
2	Beta2 Motor Dir	CW	CCW
3	Unused		
4	Unused		
5	Alpha Decode Reset	Normal	Reset
6	Beta1 Decode Reset	Normal	Reset
7	Beta2 Decode Reset	Normal	Reset

Table 13.3: HC11 Output Address Bits ($9000)

13.1.4 Inputs

Limit Switches

The limit switches can be read in on address $9002. The Table 13.4 gives the order within the byte.

Bit	Limit Switch
0	Alpha Limit a
1	Alpha Limit b
2	Beta1 Limit a
3	Beta1 Limit b
4	Beta2 Limit a
5	Beta2 Limit b
6	Unused
7	Unused

Table 13.4: Limit switch Bits ($9002)

Quadrature Decoders

The quadrature decoders are 16-bit devices. They require two consecutive reads to get the full value of the counter. The High byte is read in first, followed by reading the low byte. Thus a typical read of the Beta2 decoder would look like:

```
LDAA $900f
STTA high_byte_loc
LDAA $9007
STTA low_byte_loc
```

This loads the high byte into accumulator A, and then stores it in memory at location high_byte_loc. The same is done for the low byte.

13.2 Absolute Position

13.2.1 Limit Switches

Limit switches are needed on joints to insure the motor does not over drive the limits of the joint. It is important to protect the integrity of the motors and the joint mechanisms. A limit switch needs to be positioned at the place where the robot's limb is expected to have stopped. When the limb triggers the limit switch, the switch should stop the movement of the motor without CPU intervention and send a signal to the CPU to generate a possible fault condition. Each limb requires two limit switches; one for each maximal position. Each switch should be able to veto further movement in the direction, and only the direction, it is monitoring. If the CPU fails and a limb is in motion, the CPU will not be able to detect the position of the limb, nor will the CPU be able to stop over drive of the motor and joint. Therefore, the switches must have some control over the motor driver circuits. If the CPU has not failed, but the position sensors are not calibrated correctly because of thermal problems, such as in the case of a potentiometer, the CPU must be able to respond to the limb reaching its mechanical limit.

Design

The limit switches needed to be placed where they can be triggered by movement of the joint or limb. I chose to put the switches near the joint and trigger them with a cam attached to the gear head on the joint. This would keep wire length to a minimum. SPDT switches were chosen because this would add versatility to the debounce circuitry and CPU interface.

Further

The limit switches need to be purchased and mounted. It is suggested that push on con-

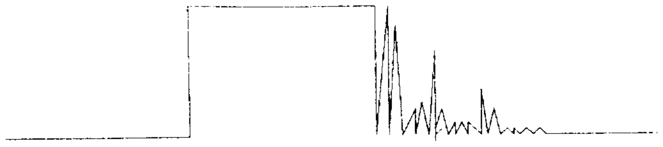


Figure 13.5: Switch Bounce

nectors be used rather than soldering to the switches. This will decrease the initial wiring time and promote ease of retiring later should it be required.

Need for De-bouncing

When a switch is closed, the contacts of the switch "bounce". That is, the two contacts actually separate and reconnect, typically 10 to 100 times over a period of 1 ms. The waveform shown in figure 13.5 is an example of the result from switch bouncing.

The output of any logic gate will faithfully respond to all those extra "pulses" caused by the bounce.

De-bouncing Circuit

The cure for switch bouncing can be found in figure 13.6.

The flip-flop changes state when the contacts first close. Further bouncing against that contact makes no difference, and the output is a "debounced" signal as shown in figure 13.7.

This debouncer circuit is widely used and can be implemented with only one 16-pin chip–a quad SR latch.

13.2.2 Potentiometers

Introduction

The potentiometer is implemented to sense the rotary position of the robot's joints by transforming the change in resistance of the potentiometer into an analog voltage level. First, the

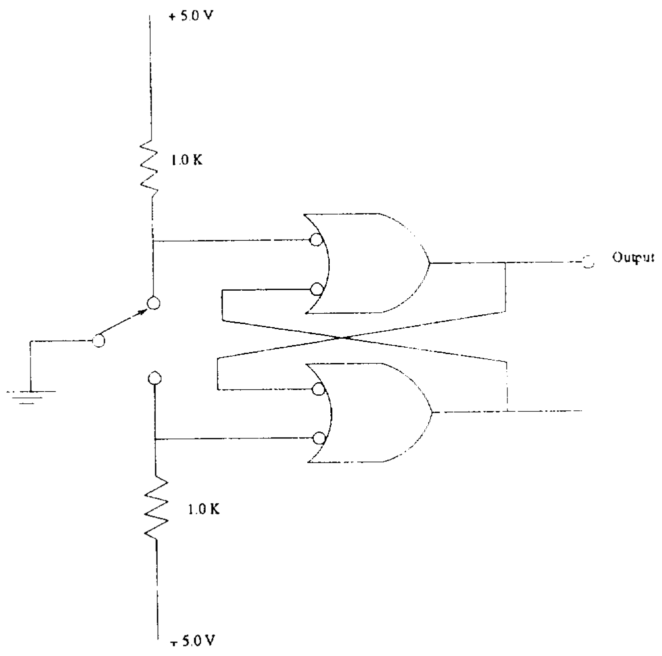


Figure 13.6: Switch Debouncer

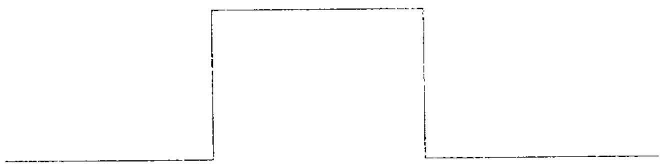


Figure 13.7: Debounced signal

selection criteria for a potentiometer will be discussed, and then its implementation with an amplifying circuit.

Selection Criteria

Two parameters of the potentiometer are of particular importance for its application.

Multiple turns: Potentiometer are typically available in 1, 3, 5, or 10 "turns". The greater number of turns allows the potentiometer to have improved resolution and linearity, both of which are of critical importance for the potentiometer's application.

Resistance Rating: The resistance of the potentiometer affects the power that the potentiometer will consume as well as its resolution. The resistance is inversely proportional to its power consumption and directly proportional to its resolution. Since there will be a total of 18 potentiometer used in the final design, low-power consumption is desired. However, a high resolution is also of critical importance, thus a trade-off has to be made.

Optimal Design

The optimal design for the potentiometer is a 10-turn, 10 KOhm potentiometer with a 1-watt power rating. The reasoning for these specifications is justified below:

- A 10-turn pot is used for high resolution and linearity.
- A 10 KOhm resistance rating is a good medium. Since the voltage is rated at a maximum of 5 volts, the current rating is 5/10E4 = 0.5 mA, which results in a power consumption of 0.25 mW, well below the potentiometer's 1-Watt power rating.
- 10 KOhm allows for good resolution and low power consumption.

Actual Design Implemented

Although a 10-turn, 10 KOhm potentiometer would have been optimal for "sensing" the position of the legs, it has been determined that an appropriate gear ratio to utilize the 10- turn potentiometer is not available. Since a one-to-one gear ratio is available, a 500 KOhm, one-turn potentiometer will be implemented. The choice for a 500 KOhm potentiometer is to accommodate the needed amplifying circuit.

Amplifying Circuit

The purpose of the amplifying circuit is to increase the maximum limited rotation of the potentiometer as if it was rotated a full 360 degrees, thus improving resolution and linearity. An amplifying circuit implemented with the potentiometer will help rectify two of the potentiometer's limitations:

- The relatively poor resolution and linearity of the one-turn potentiometer.
- The robot's joints will not travel a full 360 degrees of rotation. Thus the full range of the potentiometer will not be used and is in a sense "wasted", further degrading the resolution and linearity capabilities.

Shown below in figure 13.8 is the amplifying circuit for the potentiometer.

The circuit features a null adjust potentiometer that nulls out any DC offset at the output whenever the input signal is at zero. This is a one-time adjustment that can be made with a small, printed-circuit potentiometer.

For this circuit, two resistance values must be determined, Ri and Rf. There are a couple of guidelines for choosing these two resistance values. They are:

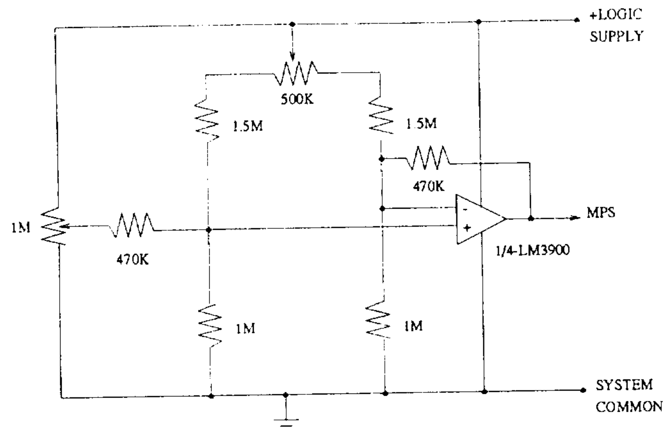


Figure 13.8: Amplifying Circuit for the Potentiometer

- *Voltage Gain.* This is the ratio of the maximum desired output voltage to the maximum input voltage. This ratio is also equal to the ratio of Rf to Ri.

- *Ri* . As a rule of thumb, the input impedance of the operational amplifier should be as high as practical. So that means Ri should be 100K or more.

The design implements Ri = 470 KOhm. This is justified by the second design criteria above.

Using a 500KOhm, one-turn potentiometer and Rf = 1 Mohm allows for a maximum of 170 degrees of rotation through the joint before the output voltage level becomes clipped. If the maximum degree of rotation is to be changed, this can be easily accomplished by changing the value of the potentiometer's maximum resistance and the value of Rf.

The potentiometer used in the null adjust , the 500 KOhm one-turn potentiometer, the 100KOhm and 500KOhm resistors, and the LM3900 op-amp are all very common parts and are readily available. The cost of this circuit is estimated to be approximately $10.

13.3 Motor Driver

13.3.1 Requirements

The necessity for a motor driver circuit arises from the need to control the voltage of the power supply using pulse-width modulation (PWM) from the processor. For this particular application, the motor driver needs to be reversible and capable of supplying 24 V at a current of 4.5 A. An 11 pin chip is manufactured by SGS-Thomson called the L6203 that can handle these requirements. The purpose of this section of the report is to explain the design of the circuit for the chip and to discuss how it interacts with the other electronic systems on the robot.

13.3.2 Design

The pin-out and block diagram for the L6203 is shown in figure 13.9. It utilizes 11 pins to form a full-bridge driver with four DMOS transistors. The enable pin turns the transistors ON and OFF when a PWM signal is applied. This mode of operation is known as enable chopping and represents the operation that is being used on the robot.

Pins labeled IN1 and IN2 are used to control the direction of motor operation. By convention, IN1 will be high (+5 V) when the desired motor rotation is clockwise. Conversely, IN2 will be high when motor rotation is counter-clockwise.

The sense pin is used for current sensing applications. It allows the placement of a resistor between the sense pin and ground that will conduct the full amount of motor current. Therefore, the voltage across the resistor will be proportional to the current through it. This voltage can be sent to the processor for use in current/force sensing applications (see Force Sensing section). One ohm resistors are used to give a voltage range of 0 to 4.5 volts for processor input.

The pin labeled Vref is for internal voltage

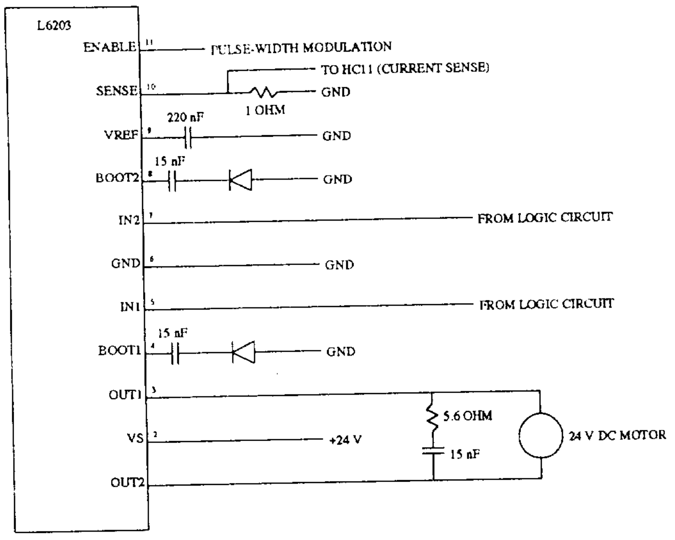


Figure 13.9: Motor Driver Schematic

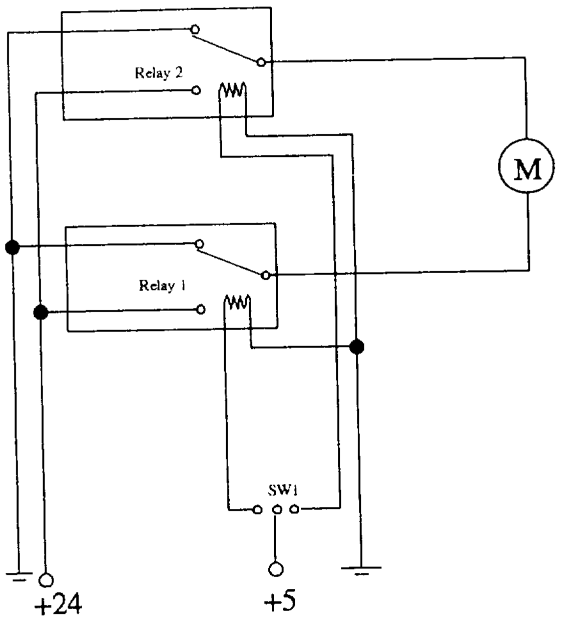


Figure 13.10: Schematic of Relay Circuit.

reference and is not used in this design. It is recommended that this pin be tied to ground through a capacitor of 220 nF.

Pins labeled BOOT1 and BOOT2 are used to allow efficient driving of the DMOS transistors using bootstrap capacitors. Capacitor values of 15 nF were selected for bootstrapping. Also, diodes were placed between the capacitors and ground in such a way to prevent the voltage of the output from dropping below ground potential.

Outputs of the power supply are connected to pins labeled Vs and GND. The chip can tolerate Vs to reach 52 volts. However, for the motors being driven, the voltage should be no more than 24 V.

The output pins labeled OUT1 and OUT2 are connected to the motor leads. The signal at these terminals will be an amplified version of the signal at the input terminals. The PWM signal will have a frequency around 1 kHz. An RC snubber circuit is placed in parallel across the output leads for filtering purposes. The values for these components were calculated from equations in the data sheet to be: R=5.6 ohms and C=15 nF.

13.4 Manual Control Panel

A control panel to manually control all of the joints of the robot was designed to run tests on the mechanics, while the electronics for the computer control were being implemented and tested separately. Design criteria for the panel were that all 18 joints must be usable in both directions, and that the power for the motors would not be run through the switches themselves. The second requirement is for safety and ease of cabling, since the motors can draw up to 5A at 24V. This design necessitates a set of relays for each motor to switch power.

The final design consisted of eighteen identical circuits (see figure 13.10). Each circuit consisted of 2 DPST Relays and 1 MON(Momentary On)/Off/MON switch. The circuit contains two separate power supplies, one for relay actuation, and another for driving the

Switch Position	Motor Terminals +	Motor Terminals -	Motion
On (Up)	+24V	Gnd	Clockwise
Off	Gnd	Gnd	Stopped
On (Down)	Gnd	+24V	Counter

Table 13.5: Switch Positions vs. Motor Direction.

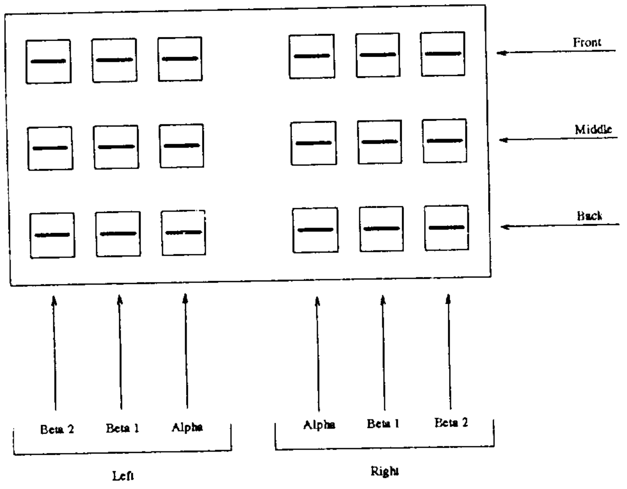


Figure 13.11: Layout of Control Panel.

motors. When the circuit is in the Off state, both of the terminals on the motor are connected to ground. (see table 13.5) Each of the On positions actuate one of the relays switching that terminal of the motor from ground to +24V. This provides a bi-directional control of the motor.

Physically the control system consisted of a control box with the 18 switches mounted as shown in figure 13.11. The relays were mounted on 2 boards, each of which controlled one side of the robot. Power was supplied by an external 24V supply, and a 5V supply.

14 Sensing

To walk across smooth level ground the robot does not need much information about its environment. With just the ability to sense the positions of each of its legs, and joints, the robot can move through preset walking patterns. As this robot was to walk up stairs and across rough ground more sensor information is needed. obstacles like stairs and walls need to be detected, and paths planned over and around them. The robot will have two methods for detecting obstacles and collisions.

Out on the legs will be reflective infra red sensors to detect nearby obstacles before collision. Secondly, motors draw more current when they have to work harder, and so if the processor watches the current the motor is drawing, it can determine if the leg has struck and obstacle. Current sensing can also be used to detect joint damage, and jammed gears, and so will be used on every motor. (see fig 14.1)

To keep the robot from tipping itself over

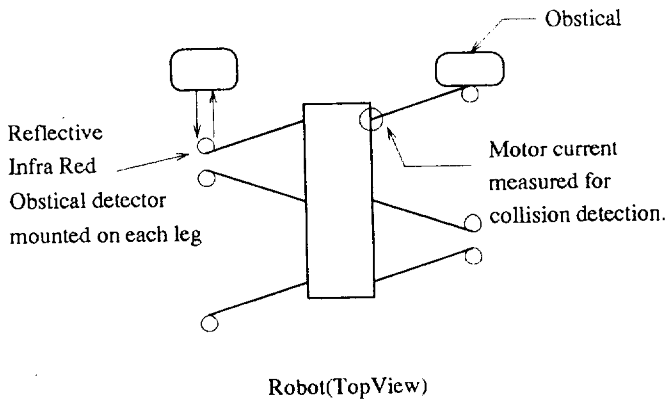


Figure 14.1: Collision Detection

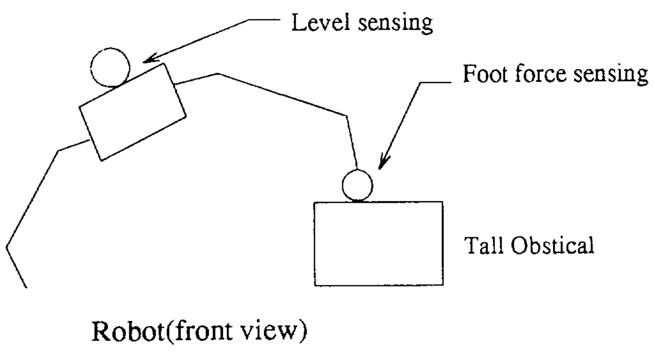


Figure 14.2: Over-balance Detection

when it steps upon a tall obstacle or down into a hole force sensors and inclinometers will bee added. (see fig 14.2) The force sensors will be mounted on the robots feet, to determine when the foot has actually touched ground, and how much of the robots weight is standing on it. The robot can tell if a foot has landed on an obstacle sooner than expected, and if the foot is holding to much weight(as will be the case if its over-balancing). The inclinometer will give the robot feedback on how much it has tipped, and the direction needed to move to compensate.

While these sensors will allow the robot to walk over most terrain as this project progresses many more sensors will need to be added, such as ultrasonic range finders, cameras, and heat sensors for detecting people.

15 Power Supply

15.1 Requirements

The requirements of the power supply are determined from the operating constraints of the motors. These constraints include a voltage of 24 V and a locked rotor current of 4.5 A per motor. Power supplies capable of producing a 24 V signal are common. However, the current of 4.5 A per motor becomes significant considering eighteen such motors will be mounted on the robot. The maximum possible current occurs if all eighteen motors are locked and energized. In this case, the current would be:

$$18 motors \times 4.5A/motor = 81A$$

However, it is unlikely that the robot would ever be in a situation where all 18 motors were locked and drawing 4.5A of current. What is more likely is that a few of the motors might be operating at near lock up, such as when climbing a flight of stairs. In this circumstance the robot would be using the six alpha motors to lift itself. If all six motors were to stall the robot would draw $6 \times 4.5A = 30A$ of current. Given this "worst case" estimate a 40 amp power supply was settled upon and acquired from the EECE department.

15.2 Future interests

An option that needs to be explored in the future involves building a supply that meets the exact requirements of the robot.

One configuration is a full-bridge rectifier with a capacitive filter. A transformer would be needed to transform 120 V ac down to 24 V ac. Thyristors could also be used to allow the dc voltage to be varied.

Part IV

Control System

16 Overview

The task of controlling a 6 legged, 18 jointed robot is complex. There are an enormous number of tasks that must occur simultaneously. Many of these tasks have equivalent priorities. Given this, it is difficult to prioritize these tasks so that they may be executed serially and still guarantee that all tasks will be serviced in a reasonable amount of time. Therefore, it was decided to use multiple processors, and divide the essential tasks among them.

The current design is one processor on each leg which is responsible for moving the three joints on that leg to given target positions. These processors will also be responsible for gathering sensor data. Between each leg pair will sit another processor that will collect this sensor data from the lower processors and make high level decisions based on this data. There will be three of these high-level processors, one for each pair of legs. These high-level processors will be capable of communicating with each other and exchanging information.

The Motorola 68HC11 micro-controller was chosen for the low-level joint manipulations and sensor data gathering. The HC11 was chosen because of it's compact size, on-board analog to digital converters, and built in pulse width modulation capabilities. It was shown that given it 2MHz clock the HC11 would be capable of performing the necessary 1KHz Pulse Width Modulation, Proportional Integral Derivative Control algorithm, sensor data gathering, and serial communications in each second in approximately on sixth of its 2,000,000 clock cycles (see Table 16.1).

Total Machine Cycles		
$50,000 \times 3$	150,000	Cycles for PID loop
	150,000	Cycles for PWM loop
	25,000	Cycles for send/rec
	325,000	Total Cycles

Table 16.1: Total machine cycles required for HC11.

The Motorola M68HC16 micro-controller was chosen for the high-level control system. It was chosen primarily for its compatibility with the HC11, its enlarged address bus, and 16MHz clock rate. The HC16 and the HC11 both have a synchronous serial communications port that is capable of baud rates approaching 2Mb/s. The HC16 also has an asynchronous line that can operate in a multi-drop mode that allows an Ethernet style protocol to be implemented. The HC16 has an address space of 1 MB in combined instruction and data mode, or 2 MB in separate instruction and data mode. This gives great flexibility in program design, and language choice, since some compactness can be sacrificed for ease of coding and readability.

17 Simulation

Due to the inherently long lead time on actual hardware for testing control code, and the expensive nature of testing code on actual hardware, a limited simulation of the robot was developed. This simulation was limited in that it only provided for joint position feedback rather than a complete complement of sensor inputs.

This simulation was used to develop and test the initial walking algorithm.

17.1 Design Criteria

This simulation was to be a static position evaluator, that took the 18 joint positions, and translated them into a 3-D wire frame of the robot. The simulation was to take commands to set new target positions for individual joints, and query the current position of any joint. There was also to be one other command that causes the simulation to increment time by some interval. As time incremented the simulation would move the joints progressively closer to their respective targets. The simulation was also required to have a lisp interface.

17.2 Implementation

The simulation is divided into three layers:

- The Lisp interface
- The C interface calls
- The robot display routines

17.2.1 Library Interfaces.

The Lisp interface is simply a set of primitives that make calls to the C interface. For more complete documentation on how to use the Lisp interface see the Appendix.

The C interface includes four main calls:

- `put_joint` which takes a leg and joint designation, and a target position, and sets the specified joint's target.
- `get_joint` which takes a leg and joint designation, and returns the current position of the specified joint. This is not necessarily the target position, but it is the actual position of the leg as time elapses.
- `update_simulation` which takes a length of time and moves the joints towards their targets.
- `init_simulation` which takes the complete dimensions of the robot (leg lengths, body width, height, and length, etc.), the joint limits, and the angular velocity of the joints. This initializes the graphics system, and the joint positions for the simulation.

17.2.2 Robot Display.

Displaying the robot was done using a 3-D wire frame modeling device independent graphics library, VOGLE. This library uses a transformation matrix, and a Cartesian coordinate space to define graphic objects.

Displaying the robot can be divided into two essential tasks:

- Displaying the legs
- Displaying the body

Displaying the legs is a relatively simple task. Since they are always to be attached to the body, they are drawn relative to the body's current orientation.

Displaying the body is a more complicated task, however. Since the legs may be in any position within their limits of motion, it is necessary to determine the orientation of the body in terms of rotation and translation matrices.

There are several steps in determining the orientation of the body:

1. Determine the points in contact with the ground
2. Determine the height from the standing plane to the body
3. Determine the standing plane's rotation about the Z axis

4. Determine the standing plane's rotation about the X axis

The method used to find the points in contact with ground was to sort a fixed number of points on the body by their vertical distance from the body. The first three points not on the same side of the body are then chosen to define the standing plane.

In order to compute the height, height must be defined. The height is the perpendicular distance from the origin to the plane. It is then a simple matter to compute using 3-D vector Analysis.

Theta, the rotation about the Z axis is computed by converting the Cartesian coordinates to spherical coordinates.

Phi, the rotation away from the Z axis is also computed by converting to spherical coordinates.

The body height transformation is then made by first translating the body by the height about the Z axis. The body rotation is done by rotating about Z by theta, then rotating about Y by phi, and then rotating the body back about Z by -theta. This places the body into the correct position so that the points in the standing plane will be on the "ground."

18 Control System

18.1 Control Background

There are many ways to control a robotic system. The most widely used method involves 3-D modelling and inverse kinematics in making decisions. This method is extremely intensive and requires large amounts of processing power. Dr. Rodney Brooks at M.I.T. has developed another approach to robotic programming called subsumption programming. The underlying assumption of this new approach is that it is rarely necessary to know everything about a given situation before any decisions are made. Therefore, a robot could "react" to certain important features of the current environment while ignoring what are deemed to be irrelevant. The burden is then placed on the programmer to then determine at what times different features are relevant or not.

18.1.1 Subsumption

The subsumption architecture was developed by Dr. Rodney Brooks at the MIT mobile robot lab. The basic premise of the architecture is that competence of the robot is built up in task achieving layers. Each layer reads sensor input and controls actuator output(eg. the motors). The lowest level layers handle the very basic, very general tasks, such as walking on level ground. The higher level competence would be aimed at more specific tasks such as walking on a slope, or climbing obstacles. When the higher level controls notice that the robot is in a position that their specialty applies, they take control away from (subsume) the lower levels. The lower levels are not stopped from running, they just no longer can send control messages to the actuators. This has the effect that if the higher level layers should fail, the lower level layers can take back over (with lessor ability, but at least the robot does not halt.) A better description of the architecture can be found in [Brooks 1986] [Brooks 1990].

18.1.2 Choices

When this project began the design team had very little experience with real time control systems. Attempting to weigh the merits of various systems was near impossible. It was decided to use the Brooks architecture as it was the most familiar, with the provision to research other systems. What follows is a listing of some of the reasons argued for subsumption.

Easily the strongest reason for choosing the subsumption architecture was that it had already been used to control two walking robots, Gengis [Brooks 1989] and Attilla, and the design team wanted to evaluate its effectiveness themselves.

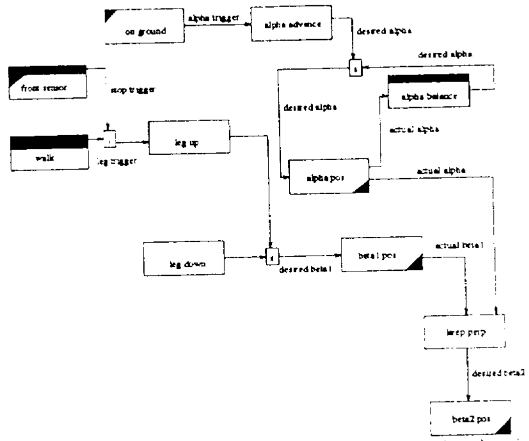
When subsumption was chosen the initial design weight of the robot was 12lbs (mostly motors) and there was not weight to spare for the mounting of large computer systems. To fit in the design goal of all on board computing, the control system had to be very efficient. Subsumption would appear to be very efficient, as one of the 6-legged walkers controlled 12 degrees of freedom with only 4 8bit, 2MHz processors, which was the kind of minimal control system that was needed.

18.1.3 Further

There is no doubt that the choice of subsumption represents a compromise and large amount of research is needed into other successful control systems. Ohio State University and Carnage Mellon University both have built walking machines. Also at MIT Colon Angle has published his masters thesis on Attilla, which the design team has not had time to acquire.

18.2 Behavior Language

The first step in designing the upper level control system was to produce an complete, initial walking plan. The basis for the design is behaviors. Behaviors are groups of rules which become activated by certain conditions within the robot. No data structures are shared between behaviors. All behaviors are asynchronous and appear to run in parallel. Each behavior controls a specific function, and only receives the information necessary for it to function from other machines. Behaviors are connected together using inhibitions and subsumptions. These connections allow one behavior to either inhibit or subsume actions of another behavior.



The boxes without bands on top are repeated six times, one for each leg. The boxes with solid bands are unique and comprise the control.
The boxes with a filled triangle in the bottom right corner control actuators.
The boxes with a filled triangle in the upper left corner receive inputs from sensors.

Figure 18.1: Subsumption Network of Behaviors for Level Ground

18.3 2-Dimension Walking Plan

The design for walking on level ground with only 2 degrees of freedom is shown in Figure 18.1.

On this diagram, each box represents a behavior. Those boxes without bands on top are copied six times, once for each leg. Those with filled triangles in the lower corner actually control the legs. Those with solid bands are the "central control" of the robot. Those with triangles in the upper corner get their inputs from the sensors.

18.3.1 Walking Plan

The behaviors shown in Figure 18.1 work together to form a simple walking plan for level ground. Control is achieved by connecting the machines and using inhibitions and subsumptions. Not all of this walking plan was implemented due to time constraints and the lack of sensor data. Therefore, a two-dimensional walking system using only joint positions was implemented. That which was implemented is described below, and the behavior language code for it is found in the Appendix.

1. **Standing**. There are three behaviors which control the lowest level of control and interface with the motor control loop. *Alpha pos, beta-1 pos* , and *beta-2 pos* receive a desired position, outputs that position to the motor, and returns the actual position of the joint. There are given positions to which the joints will go when the robot is powered up so that it stands.

2. **Leg lifting**. A *leg down* machine will always output a position to the *beta-1 pos* which will place the leg in the down position. Another machine, *up-leg-trigger* subsumes the *leg down* machine when the leg is raised to walk.

3. **Leg swinging**. There is a single machine, called *alpha balance* which accepts the alpha positions from each of the legs. It then sums the alpha position, where 0 is straight out, positive is forward, and negative is backward. Based on the sum, *alpha balance* outputs a signal to each of the *alpha pos* machines which will adjust the legs to keep the body centered. If one leg moves forward, *alpha balance* will move all others backward to compensate.

   For each leg, there is an *alpha advance* machine. Whenever the leg has been raised, it suppresses the *alpha balance* output to that leg, and outputs a position to the *alpha pos* machine which will swing the leg forward. So when *up-leg-trigger* raises a leg, *alpha advance* swings it forward.

4. **Walking**. Finally, a behavior must be added to trigger the *up- leg-trigger* machines in the appropriate order to produce the desired gait. For walking on level ground, the *walk* machine sends out triggers to implement the tripod gait.

18.3.2 Behaviors

Each behavior is described with its inputs, outputs and functions below.

- **Alpha-pos** There are six of these machines, one for each leg. Input to this machine is *signal-pos*, which is the position which the other machines signal they would like the alpha joint to be placed. The output from *alpha pos* is *current-pos* which is the actual position of the alpha joint. Its function is to interact with the lower level control loop to move the motor to *signal-pos*, and to get *current-pos* from the lower level and output it for other machines to use.

- **Beta-pos** There are six of these machines, one for each leg. Inputs outputs, and functions are exactly the same as those in *Alpha-pos* except that they control the beta-1 joint.

- **Alpha-advance** Again, there are six copies of this behavior, one for each leg. Once the robot has stood up, each *alpha-advance* machine receives a *go* input. After that, they check to see if the *beta-pos* input says that the leg has been raised. If so, the machine outputs a new *alpha-pos* to swing the leg forward.

- **Leg-down** The six copies of this machine have a simple function. As input, they take *current-beta*. If this position is not the same as the *leg-down-pos*, a constant, it outputs the new beta position in *new-beta* to put the leg down.

- **Up-leg-trigger** This machine, one on each leg, takes as input a *trig* or trigger which indicates that this leg should be raised. When the trigger has been received, the machine outputs *new-beta* to raise the leg.

- **Alpha-balance** There is only one of this machine. It takes as input the alpha positions of all six legs. Once all six have been received, it sums them to determine whether they should be altered. If the sum is greater than 0, a constant amount is subtracted from each *alpha pos.* Likewise, if sum is less than 0, a constant is added to each *alpha pos.* Output from this are all six new alpha positions.

- **Walk** This machine sends out the signals to the legs in the proper order to obtain the tripod gait. It contains monostables (timers) to keep track of which set of legs was last triggered. Once the *go* input has been received, it outputs the appropriate of the six triggers.

- **Stand** This machine makes sure that at least three of the legs are on the ground before it begins to walk. It takes as input the beta-1 positions of the six legs, and outputs a *go* signal, if appropriate.

The behavior machines interact with each other by connecting the inputs and outputs of the machines with the *connect* statement. Connections which were used in the implementation were :

1. The output *current-pos* from beta-pos is connected to the input *beta* from stand for each of the six legs.

2. The output *go* from stand is connected to the *go* input from the walk machine, and also to each of the *go* inputs for the alpha-advance machines.

3. The outputs *signal* for each leg from alpha-balance are connected to the *signal-pos* for each of the alpha-pos machines.

4. The output *current-pos* from each of the alpha-pos machines are connected as inputs to the alpha-balance machine and to the alpha-advance machines.

5. The output *current-pos* from each of the beta-pos machines are connected as inputs to each of the alpha-advance machines.

6. The output *alpha-pos* from the alpha-advance machines are connected to the input *signal-pos* for the alpha-pos machines. These outputs also subsume all other input to alpha-pos.

7. The outputs *new-beta* from the leg-down machines are connected to the inputs *signal-pos* for the beta-pos machines.

8. The outputs *current-pos* from the beta-pos machines are connected to the inputs *current-beta* for the leg-down machines.

9. The outputs *new_beta* from the up-leg-trigger machines are connected to the inputs *signal-pos* for the beta-pos machines. Also, these outputs subsume all other input to beta-pos.

10. The outputs *trig* from the walk machine are connected to the inputs *trig* for each of the up-leg-trigger machines.

18.4 Other work

The two degree of freedom, joint position controlled implementation described above was tested on the animation program which was designed. The behavior code was translated into common lisp code, and then run on top of the animator. From this testing, we determined that this implementation moves legs in the correct sequence and with the right timing to walk. The stand procedure was also working when the animator was started.

18.5 Further

The calculations for beta-2 need to be implemented into the behavior *keep-perp* so that testing can be done on the animator to check that indeed that joint is keeping the leg perpendicular at all times. Also, the implementation will have to be updated to accept sensor input from the other machines. This includes writing the behaviors to handle the sensor data and changing the existing behaviors to use the sensor data. For example, the walk behavior should be changed so that when the front sensors detect an object, the triggers are no longer sent out and the robot stops walking until a suitable evaluation of the object which was detected has been done.

19 Processor Communication

19.1 Operating Systems

In the robot there are two types of processors. There are 3 Motorola MC68HC16 processors and 6 Motorola MC68HC11 processors. The two types of processors do completely different jobs, and require 2 separate operating systems. The processors are organized in three sets. Each set consists of a HC16 connected to two HC11 processors. The sets are connected between the HC16 processors only.

19.1.1 MC68HC16

On the MC68HC16 processors there are three components to its operating system: A system clock, an inter-HC16 network driver, and a network driver for the HC11 $\Longleftrightarrow$ HC16.

The system clock is handled by the periodic interrupt of the HC16, it is to be incremented every other interrupt. It is used to time-stamp data, and using the time-stamps invalidate old data.

The HC16 $\Longleftrightarrow$ HC16 network driver consists of an interrupt routine that monitors the asynchronous serial device for incoming packets. These packets are parsed by the routine and acted upon. Most packets will be data register transfers, and are quickly handled. There are to be provisions for command packets such as a shutdown packet that will halt the processor.

The HC11 $\Longleftrightarrow$ HC16 network driver is keyed off of the periodic interrupt, on each interrupt it will query one of the HC11 processors for data. This in conjunction with the behavior of system clock means that the data of both HC11 processors connected to the HC16 will be updated on each tick of the system clock.

19.1.2 MC68HC11

On the MC68HC11 processors there is only one component to its operating system. This is the communications driver that monitors traffic from the HC16 processor. This interrupt is triggered externally by the HC16, so if the HC16 processor loses communications for some reason, the HC11 will continue to move the motors until they reach their currently desired positions and halt.

19.2 Network Subsystem

19.2.1 HC11 $\Longleftrightarrow$ HC16

Each HC16 is connected to two HC11 processors via its synchronous serial interface. Data transfer between each HC16 and its two HC11 sub-processors will be automatic and will occur 30 times a second for each HC11. All serial routines on every processor will be interrupt driven. Data transmission will be single buffered, and all data reception will be double buffered.

Data transfer from each HC11 to the HC16 will consist of the 3 joint encoder values, and the 8 A/D converter values. These are organized into a 16 byte packet. The data packet will contain a STX, then 3 16-bit encoder values, 8 8-bit A/D values, and an ETX.

Data transfer from the HC16 to each HC11 will consist of the 3 desired joint encoder positions. Each joint position will be transmitted twice for error recovery. The data packet will be the same length in both directions on the serial link (16 bytes), and will be organized as follows: STX, 3 16-bit encoder values, NUL, NUL, 3 16-bit encoder values, ETX. On reception the duplicated encoder values will be compared, if they are not equal the encoder value closest to the current position will be used.

Since packet lengths are 16 bytes, and each byte is about 10 bits to transmit, there are 160 bits per packet. There are to be 30 packets transferred each second, which makes 4800 bits/second. Since the two HC11s cannot be talked to at the same time this value doubles to 9600 bits/second maximum data transfer. The HC11 processor is clocked at 2MHz, which using a div4 clock rate on the serial system yields 524288 bits/second. Therefore there is more than enough bandwidth to transmit the necessary data.

19.2.2 HC16 $\Longleftrightarrow$ HC16

The three HC16 processors are connected using their asynchronous serial interfaces in multidrop mode. This mode makes the serial interface look similar to an Ethernet interface. A CSMA-CD (Carrier Sense Multiple Access - Collision Detect) style protocol will be used for packets.

The HC16 network is to be used to send data between processors. The design calls is for data to be updated across the network as it is updated on the local processor, each data value will be time-stamped as it arrives. This provides a sort of shared memory segment between the three processors. A data transfer will consist of a STX, A port number(address), and a 16 bit data value. The port number will be a 16 bit value, which makes the packet size 5 bytes. Special port numbers will be assigned to allow command type functions to occur between processors. There are no acknowledge packets, each processor will monitor read its own outgoing packets and compare them to what was to be sent. If the incoming packet was different from the outgoing packet, the send will fail, and the packet will have to be resent. This can either be done be the control program, or by a library routine wrapper around the send function. This option exists because a delay must be introduced to avoid network deadlocks, since most errors will be caused by multiple processors transmitting simultaneously.

19.3 Program Interface

There are 3 functions in the interface specification for user programs: SetPort, LoadPort, and ValidPort. SetPort sets a given port to a given value on all processors on the network. LoadPort returns a given port's value from the local computers storage. ValidPort tests to see if a given port's data has not timed out.

19.3.1 SetPort

Prototype: `int SetPort(Port port, int data)`

SetPort sends a data port's value across the network. It will send the data until the network send succeeds. It does not set the local processors data port storage, this is done by the network reception interrupt routine on the processor. SetPort will test to see if the local value is correct after a network send succeeds. If the values do not match it attempts to resend the data until the local data matches the data that was to be sent.

19.3.2 LoadPort

Prototype: `int LoadPort(Port port)`

LoadPort returns the value of the given port as it is on the local host.

19.3.3 ValidPort

Prototype: `int ValidPort(Port port, int validtime)`

ValidPort returns 1 if the difference between the time-stamp on the port and the system clock is within validtime. Otherwise the routine returns 0.

19.3.4 GetClock

Prototype: `int GetClock()`

GetClock returns the current system clock tick.

20 Motor Control Loop

20.1 Motor Speed Control

20.1.1 The Problem

Ideally the way to control the speed of a motor would be to vary the voltage of the power source while letting the motor draw as much current as it needs. One way to do this would be to use a large variable resistor in series with the motor. By adjusting the resistor the voltage drop across the motor could be changed. This is very inefficient, as any excess voltage is dissipated across the resistor as heat. Another poor method of speed control would be to use a power transistor in its linear region (as an electronic variable resistor). Again any excess voltage is wasted as heat, so this solution is unacceptable.

20.1.2 A Better Solution

If the voltage to the motor could be turned on and off very quickly (say half the time on, half off), the voltage would average to be half maximum.(see figure 20.1) Suppose that a speed like one third were needed. The voltage could be switched on only one third the time off the rest, averaging to one third max voltage. This technique is called pulse width modulation. It turns out to be rather efficient, as the power transistors driven from totally on to totally off(and vis versa) behave like ideal switches and dissipate very little power. Ideal switches have no power dissipation because when they are off, no current flows, so the power equation equals zero.

$$Power = Voltage \times Current$$

When the switch is on, and current is flowing the ideal switch has a zero voltage drop, again the power equation equals zero. The DMOS power transistors we are using are not ideal switches(they have some voltage drop when they are on), but are fairly close.

20.1.3 Requirements

The pulse width modulation code has 3 main subsections. The first is the subsection called by the PID program (called the mainline section). This subsection allows the PID routine to change the requested speed of any motor. It will be implemented as 3 assembly functions (speeda, speedb1, speedb2), one function per joint motor, alpha, beta1, and beta2. Each function will take a percentage of the maximum speed, and a motor direction as arguments. The motor direction commands to the motor drivers will be sent out through a latch which is mapped into the memory (see section 13.1)

The second section will look at the requested speeds and output the proper signals to the motor drivers. This subsection has strict timing requirements, and so is implemented with a hardware interrupt (and is called the interrupt subsection). The PWM signals to the motor drivers are sent out of the 68hc11s PORT A (the timing port).

The third subsection will be the initialization subsection that is run when the processor is reset, and it is responsible for clearing all variables, and initializing the interrupt routine (called the initialization subsection). The most stringent requirement that must be held by all subsections is that the PWM code must be extremely time efficient. It has only 150,000 cycles per second to operate.

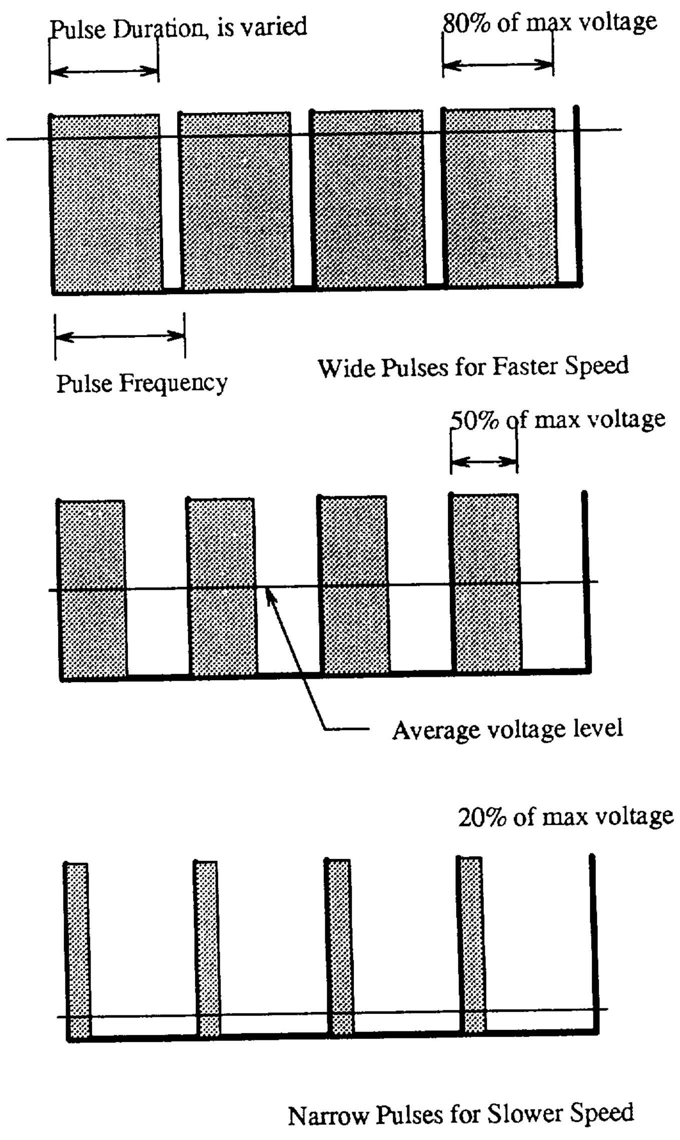


Figure 20.1: Pulse Width Modulation.

20.1.4 Implementation Details

The mainline functions are simple in concept, first they take the motor direction, and if it has changed from previously, write new direction bits to the motor direction latch. Second, they take the percentage of full speed and convert that value into an equivalent number of clock cycles and put that value in the PwmLen1,2,3 variable(depending on which motor).

The interrupt subsection is a little more complex, and relies on the operating systems main timer. The main timer will be an interrupt (OC1) that is triggered once per millisec(1000Hz). The interrupt subsection of the PWM is to be part of this main timer (so it runs 1000 times a second). To insure fast and accurate control of the control pulses, the interrupt routine is to use some of the HC11's built in timing systems. [For a detailed discussion of the timing subsystem of the 68HC11 processor, see chapter 10, section 4. reference hc11 handbook]. The PWM code will be a direct extension of example 10-6 *OC1, OC2, and OC3 used together to produce 2 PWM signals* Essentially how the timing system works is that a motor control output is set high and a time delay is programmed into one of the timer registers. When the time delay is up, the hardware automatically toggles the control output low. The interrupt routine will just raise the motor controller output pins and program the pulse widths (already calculated by the mainline section) into the timers and then exit. The timers will delay the programmed amount, and then automatically end the pulses.

The initialization section just has the job of clearing all variables used in the PWM calculations to zero every time the processor is reset.

20.2 P.I.D. Control

PID control is one of the best-known controllers used in practice. One reason why PID control is so popular is the amount of freedom it introduces into a control system. Three constants designated Kp, Ki, and Kd represent the amount of each type of control in the system. The three types are positional, integral, and derivative (PID). The block diagram of a PID control system is shown in figure 20.2

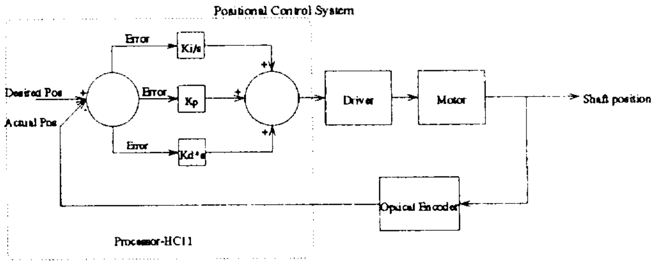


Figure 20.2: Positional Control System

By changing the values of constants Kp, Ki, and Kd, the individual amounts of each control are changed. Using root-locus or bode plots allows the designer to find values of each constant that results in a desired overshoot, rise time, settling time.

Since specific parameters such as time constants for our motors are not known, another method is available for determining these constants. Using potentiometers, the individual constants can be varied and tested until the motor behaves in the most desirable manner. Desirable characteristics include minimal overshoot, minimal rise time, and minimal settling time. Once this behavior has been reached, the settings of the potentiometers can be measured and these values become the constants in the software.

Optical encoders are being used for feedback information on the position of the motor. This information will be in the form of pulses that must be decoded to give absolute position information. The equation used for implementing PID control in software given these feedback pulses is shown below.

References

Brooks, Rodney A. 1986. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, **RA-2**(April), 14–23.

Brooks, Rodney A. 1989. A Robot that Walks: Emergent Behavior from a Carefully Evloved Network. *Neural Computation*, **1**(2), 253–262.

Brooks, Rodney A. 1990. *The Behavior Language; User's Guide*. Tech. rept. 1227. MIT AI Lab Memo.

Faires, Virgil Moring. 1965. *Design of Machine Elements*. Fourth edn. Toronto: The MacMillan Company.

Appendix A Lisp Interface to Animation

A.1 General Use

The lisp interface to the Animation is quite straight forward. To load all of the object files, and lisp code simply place `(load ~/lisp/simulation.lisp)` in your lisp code prior to using any of the functions. This will load the graphics drivers and the animation system, as well as define the lisp interfaces to be used to configure the system. After everything is loaded, `simulation.lisp` will set up and initialize the graphics window with the joint limits for the robot. At this point, you may begin issuing commands to the animation to change joint positions, request positions, or update the screen.

A.2 Animation Calls

The available calls are listed below, as well as documented online, I believe.

`(put-joint! leg joint position)` Takes three arguments the leg, the joint (`alpha, beta-1, or beta-2`), and the desired position in floating point degrees. This will tell the animation to set this position as the desired position and increment toward that position as time is incremented by `update-simulation`.

`(get-joint? leg joint)` Takes a leg and a joint (`alpha, beta-1, beta-2`) and returns the *current* position of the joint. *It does not return the most recent put-joint! for that joint and leg, but the actual position of the joint.*

`(update-simulation delta-time)` Takes the amount of time (in seconds) that has passed and moves the joints towards their respective target values, based on a set rotational velocity (this is set in the initialization in `~/lisp/simulation.lisp`).

`(close-down-simulation)` will close down the simulation and close up the graphics system. (*Note:* This will not remove the graphics window, the graphics window, will be removed when the lisp process is exited. Another simulation run may be performed while this window still exists, but another window will be created, the old one will not be used.)

`(initialize-simulation ...)` This call has a large number of arguments:

- `alpha-minimum, alpha-maximum` alpha joint limits.
- `beta-1-minimum, beta-1-maximum` beta-1 joint limits.
- `beta-2-minimum, beta-2-maximum` beta-2 joint limits.
- `angular-velocity` the velocity with which the joints are moved.
- `total-mass` the total mass of the robot.
- `upper-leg-length, middle-leg-length, lower-leg-length` the length of the respective leg segments.
- `leg-thickness` the leg tubing is square with this dimension.
- `body-width` is the width of the body from the left to right.
- `body-length` is the length of the body from front to back.
- `body-height` is the height of the body segment, not the height off of the ground.

A.3 Animation Constants

There are several symbolic values available to avoid using integers to indicate joints and legs:

`right-front, right-middle, right-rear, left-front, ...` are constants that represent the leg positions. Left is your left as you look into the monitor, front is into the monitor (i.e. the robot is facing away from the keyboard.)

`alpha, beta-1, and beta-2` are the joint positions.

A.4 Joint Positions

Joint angle positions are given in floating point degrees, 0.0 degrees is straight out from the robot for alpha and beta-1, 0.0 degrees for beta-2 is straight out from the middle leg segment. The positive direction for the alpha joints is toward the head (into the monitor), and downward for the beta-1 and beta-2 joints. The current limit for the alpha joint is ±30 degrees; the limit for beta-1 is ±90 degrees, and the limit for beta-2 is ±90 degrees. These limits are synthetic, in that they are completely changeable by editing the `initialize-simulation` call in the `~/lisp/simulation.lisp` file. All of the other "flexible" parameters of the simulation are set in this call (e.g. body-part lengths, angular velocity, total mass, etc.).

Appendix B Behavior Language Code for 2-D model

```
(defconstant leg_down_pos 45.0)
(defconstant upleg_pos 10.0)

(defbehavior update
  :processes ((whenever t
                 (update-simulation 0.04))))

(defbehavior alpha-pos-1
  :inputs (signal_pos)
  :outputs (current_pos)
  :processes (
               (whenever (received? signal_pos)
                  (put-joint! right-front alpha signal_pos))
               (whenever t
                  (output current_pos (get-joint? right-front alpha)))))

(defbehavior alpha-pos-2
  :inputs (signal_pos)
  :outputs (current_pos)
  :processes (
               (whenever (received? signal_pos)
                  (put-joint! left-front alpha signal_pos))
               (whenever t
                  (output current_pos (get-joint? left-front alpha)))))

(defbehavior alpha-pos-3
  :inputs (signal_pos)
  :outputs (current_pos)
  :processes (
               (whenever (received? signal_pos)
                  (put-joint! right-middle alpha signal_pos))
               (whenever t
                  (output current_pos (get-joint? right-middle alpha)))))

(defbehavior alpha-pos-4
  :inputs (signal_pos)
  :outputs (current_pos)
  :processes (
               (whenever (received? signal_pos)
                  (put-joint! left-middle alpha signal_pos))
               (whenever t
                  (output current_pos (get-joint? left-middle alpha)))))

(defbehavior alpha-pos-5
  :inputs (signal_pos)
  :outputs (current_pos)
  :processes (
               (whenever (received? signal_pos)
                  (put-joint! right-rear alpha signal_pos))
               (whenever t
                  (output current_pos (get-joint? right-rear alpha)))))

(defbehavior alpha-pos-6
  :inputs (signal_pos)
  :outputs (current_pos)
  :processes (
               (whenever (received? signal_pos)
                  (put-joint! left-rear alpha signal_pos))
               (whenever t
                  (output current_pos (get-joint? left-rear alpha)))))

(defbehavior beta-pos-1
  :inputs (signal_pos)
  :outputs (current_pos)
  :processes (
               (whenever (received? signal_pos)
                  (put-joint! right-front beta-1 signal_pos))
               (whenever t
                  (output current_pos (get-joint? right-front beta-1)))))

(defbehavior beta-pos-2
  :inputs (signal_pos)
  :outputs (current_pos)
  :processes (
               (whenever (received? signal_pos)
```

```
                  (put-joint! left-front beta-1 signal_pos))
               (whenever t
                  (output current_pos (get-joint? left-front beta-1)))))

(defbehavior beta-pos-3
  :inputs (signal_pos)
  :outputs (current_pos)
  :processes (
               (whenever (received? signal_pos)
                  (put-joint! right-middle beta-1 signal_pos))
               (whenever t
                   (output current_pos (get-joint? right-middle beta-1)))))

(defbehavior beta-pos-4
  :inputs (signal_pos)
  :outputs (current_pos)
  :processes (
               (whenever (received? signal_pos)
                  (put-joint! left-middle beta-1 signal_pos))
               (whenever t
                   (output current_pos (get-joint? left-middle beta-1)))))

(defbehavior beta-pos-5
  :inputs (signal_pos)
  :outputs (current_pos)
  :processes (
               (whenever (received? signal_pos)
                  (put-joint! right-rear beta-1 signal_pos))
               (whenever t
                  (output current_pos (get-joint? right-rear beta-1)))))

(defbehavior beta-pos-6
  :inputs (signal_pos)
  :outputs (current_pos)
  :processes (
               (whenever (received? signal_pos)
                  (put-joint! left-rear beta-1 signal_pos))
               (whenever t
                  (output current_pos (get-joint? left-rear beta-1)))))

(defbehavior alpha-advance-1
  :inputs (go old_alpha beta_pos)
  :outputs (alpha_pos)
  :processes ((whenever (received? go)
(whenever (received? beta_pos)
                (if (< beta_pos leg_down_pos)
                   (output alpha_pos (+ old_alpha 5.0)))))))

(defbehavior alpha-advance-2
  :inputs (go old_alpha beta_pos)
  :outputs (alpha_pos)
  :processes ((whenever (received? go)
        (whenever (received? beta_pos)
                (if (< beta_pos leg_down_pos)
                   (output alpha_pos (+ old_alpha 5.0)))))))

(defbehavior alpha-advance-3
  :inputs (go old_alpha beta_pos)
  :outputs (alpha_pos)
  :processes ((whenever (received? go)
              (whenever (received? beta_pos)
               (if (< beta_pos leg_down_pos)
                  (output alpha_pos (+ old_alpha 5.0)))))))

(defbehavior alpha-advance-4
  :inputs (go old_alpha beta_pos)
  :outputs (alpha_pos)
  :processes ((whenever (received? go)
       (whenever (received? beta_pos)
                (if (< beta_pos leg_down_pos)
                   (output alpha_pos (+ old_alpha 5.0)))))))

(defbehavior alpha-advance-5
  :inputs (go old_alpha beta_pos)
  :outputs (alpha_pos)
  :processes ((whenever (received? go)
       (whenever (received? beta_pos)
                (if (< beta_pos leg_down_pos)
                   (output alpha_pos (+ old_alpha 5.0)))))))

(defbehavior alpha-advance-6
  :inputs (go old_alpha beta_pos)
  :outputs (alpha_pos)
  :processes ((whenever (received? go)
       (whenever (received? beta_pos)
                (if (< beta_pos leg_down_pos)
                    (output alpha_pos (+ old_alpha 5.0)))))))

(defbehavior leg_down-1
  :inputs (current_beta)
  :outputs (new_beta)
  :processes ((whenever (received? current_beta)
                (if (not (= current_beta leg_down_pos))
                   (output new_beta leg_down_pos)))))

(defbehavior leg_down-2
  :inputs (current_beta)
  :outputs (new_beta)
  :processes ((whenever (received? current_beta)
                (if (not (= current_beta leg_down_pos))
                   (output new_beta leg_down_pos)))))

(defbehavior leg_down-3
  :inputs (current_beta)
  :outputs (new_beta)
  :processes ((whenever (received? current_beta)
                (if (not (= current_beta leg_down_pos))
                   (output new_beta leg_down_pos)))))

(defbehavior leg_down-4
  :inputs (current_beta)
  :outputs (new_beta)
  :processes ((whenever (received? current_beta)
                (if (not (= current_beta leg_down_pos))
                   (output new_beta leg_down_pos)))))

(defbehavior leg_down-5
  :inputs (current_beta)
  :outputs (new_beta)
  :processes ((whenever (received? current_beta)
                (if (not (= current_beta leg_down_pos))
                   (output new_beta leg_down_pos)))))

(defbehavior leg_down-6
  :inputs (current_beta)
  :outputs (new_beta)
  :processes ((whenever (received? current_beta)
                (if (not (= current_beta leg_down_pos))
                   (output new_beta leg_down_pos)))))
```

```
(defbehavior up_leg_trigger-1
  :inputs (trig)
  :outputs (new_beta)
  :processes ((whenever (received? trig)
                (output new_beta upleg_pos))))

(defbehavior up_leg_trigger-2
  :inputs (trig)
  :outputs (new_beta)
  :processes ((whenever (received? trig)
                (output new_beta upleg_pos))))

(defbehavior up_leg_trigger-3
  :inputs (trig)
  :outputs (new_beta)
  :processes ((whenever (received? trig)
                (output new_beta upleg_pos))))

(defbehavior up_leg_trigger-4
  :inputs (trig)
  :outputs (new_beta)
  :processes ((whenever (received? trig)
                (output new_beta upleg_pos))))

(defbehavior up_leg_trigger-5
  :inputs (trig)
  :outputs (new_beta)
  :processes ((whenever (received? trig)
                (output new_beta upleg_pos))))

(defbehavior up_leg_trigger-6
  :inputs (trig)
  :outputs (new_beta)
  :processes ((whenever (received? trig)
                (output new_beta upleg_pos))))

(defbehavior alpha-balance
  :inputs (a1 a2 a3 a4 a5 a6)
  :decls ((f1 :init nil) (f2 :init nil) (f3 :init nil) (f4 :init nil)
  (f5 :init nil) (f6 :init nil))
  :outputs (s1 s2 s3 s4 s5 s6)
  :processes (
                  (whenever t
                     (if (and f1 f2 f3 f4 f5 f6)
                          (sequence
                           (setf f1 nil) (setf f2 nil) (setf f3 nil)
                           (setf f4 nil) (setf f5 nil) (setf f6 nil)
                           (setf sig (+ a1 a2 a3 a4 a5 a6))
                           (if (> sig 0.0)
                              (setf signal -5.0)
                              (if (< sig 0.0)
                                 (setf signal 5.0)
                                 (setf signal 0.0)))
                           (output s1 (+ a1 signal))
                           (output s2 (+ a2 signal))
                           (output s3 (+ a3 signal))
                           (output s4 (+ a4 signal))
                           (output s5 (+ a5 signal))
                           (output s6 (+ a6 signal)))))
                   (whenever (received? a1) (setf f1 t))
   (whenever (received? a2) (setf f2 t))
   (whenever (received? a3) (setf f3 t))
   (whenever (received? a4) (setf f4 t))
   (whenever (received? a5) (setf f5 t))
   (whenever (received? a6) (setf f6 t))))

(defbehavior walk
  :inputs (go)
  :decls ((triad1 :monostable 7.5) (triad2 :monostable 7.5)
  (wait :monostable 10.0) (flag1 :init 0) (flag2 :init 0)
  (wait_flag :init 1))
  :outputs (t1 t2 t3 t4 t5 t6)
  :processes (
               (whenever (received? go)
                  (exclusive
                     (whenever triad1
                        (setf wait_flag 0)
                        (setf flag1 1)
                        (setf flag2 0)
                        (output t1 1)
                        (output t4 1)
                        (output t5 1))
             (whenever triad2
       (setf wait_flag 0)
```

```
                        (setf flag1 0)
      (setf flag2 1)
                        (output t2 1)
                        (output t3 1)
                        (output t6 1))
                     (whenever (and (not triad1) (not triad2) (not wait))
                        (if (= wait_flag 1)
                           (if (= flag1 1)
      (trigger triad2)
    (trigger triad1))
 (trigger wait)))
                     (whenever wait
      (setf wait_flag 1))))))

(defbehavior stand
  :inputs (b1 b2 b3 b4 b5 b6)
  :outputs (go)
  :processes ((whenever t
                 (setf num_down 0)
                 (if (= b1 leg_down_pos)
                    (setf num_down (+ 1 num_down)))
                 (if (= b2 leg_down_pos)
                    (setf num_down (+ 1 num_down)))
                 (if (= b3 leg_down_pos)
                    (setf num_down (+ 1 num_down)))
                 (if (= b4 leg_down_pos)
                    (setf num_down (+ 1 num_down)))
                 (if (= b5 leg_down_pos)
                    (setf num_down (+ 1 num_down)))
                 (if (= b6 leg_down_pos)
                    (setf num_down (+ 1 num_down)))
                 (if (>= num_down 3)
                    (output go 1)))))

(connect (beta-pos-1 current_pos) (stand b1))
(connect (beta-pos-2 current_pos) (stand b2))
(connect (beta-pos-3 current_pos) (stand b3))
(connect (beta-pos-4 current_pos) (stand b4))
(connect (beta-pos-5 current_pos) (stand b5))
(connect (beta-pos-6 current_pos) (stand b6))

(connect (stand go) (walk go))
(connect (stand go) (alpha-advance-1 go))
```

```
(connect (stand go) (alpha-advance-2 go))
(connect (stand go) (alpha-advance-3 go))
(connect (stand go) (alpha-advance-4 go))
(connect (stand go) (alpha-advance-5 go))
(connect (stand go) (alpha-advance-6 go))

(connect (alpha-balance s1) (alpha-pos-1 signal_pos))
(connect (alpha-balance s2) (alpha-pos-2 signal_pos))
(connect (alpha-balance s3) (alpha-pos-3 signal_pos))
(connect (alpha-balance s4) (alpha-pos-4 signal_pos))
(connect (alpha-balance s5) (alpha-pos-5 signal_pos))
(connect (alpha-balance s6) (alpha-pos-6 signal_pos))

(connect (alpha-pos-1 current_pos) (alpha-balance a1))
(connect (alpha-pos-2 current_pos) (alpha-balance a2))
(connect (alpha-pos-3 current_pos) (alpha-balance a3))
(connect (alpha-pos-4 current_pos) (alpha-balance a4))
(connect (alpha-pos-5 current_pos) (alpha-balance a5))
(connect (alpha-pos-6 current_pos) (alpha-balance a6))

(connect (beta-pos-1 current_pos) (alpha-advance-1 beta_pos))
(connect (beta-pos-2 current_pos) (alpha-advance-2 beta_pos))
(connect (beta-pos-3 current_pos) (alpha-advance-3 beta_pos))
(connect (beta-pos-4 current_pos) (alpha-advance-4 beta_pos))
(connect (beta-pos-5 current_pos) (alpha-advance-5 beta_pos))
(connect (beta-pos-6 current_pos) (alpha-advance-6 beta_pos))

(connect (alpha-pos-1 current_pos) (alpha-advance-1 old_alpha))
(connect (alpha-pos-2 current_pos) (alpha-advance-2 old_alpha))
(connect (alpha-pos-3 current_pos) (alpha-advance-3 old_alpha))
(connect (alpha-pos-4 current_pos) (alpha-advance-4 old_alpha))
(connect (alpha-pos-5 current_pos) (alpha-advance-5 old_alpha))
(connect (alpha-pos-6 current_pos) (alpha-advance-6 old_alpha))

(connect (alpha-advance-1 alpha_pos) ((suppress (alpha-pos-1 signal_pos))))
(connect (alpha-advance-2 alpha_pos) ((suppress (alpha-pos-2 signal_pos))))
(connect (alpha-advance-3 alpha_pos) ((suppress (alpha-pos-3 signal_pos))))
(connect (alpha-advance-4 alpha_pos) ((suppress (alpha-pos-4 signal_pos))))
(connect (alpha-advance-5 alpha_pos) ((suppress (alpha-pos-5 signal_pos))))
(connect (alpha-advance-6 alpha_pos) ((suppress (alpha-pos-6 signal_pos))))

(connect (leg_down-1 new_beta) (beta-pos-1 signal_pos))
(connect (leg_down-2 new_beta) (beta-pos-2 signal_pos))
```

```
(connect (leg_down-3 new_beta) (beta-pos-3 signal_pos))
(connect (leg_down-4 new_beta) (beta-pos-4 signal_pos))
(connect (leg_down-5 new_beta) (beta-pos-5 signal_pos))
(connect (leg_down-6 new_beta) (beta-pos-6 signal_pos))

(connect (beta-pos-1 current_pos) (leg_down-1 current_beta))
(connect (beta-pos-2 current_pos) (leg_down-2 current_beta))
(connect (beta-pos-3 current_pos) (leg_down-3 current_beta))
(connect (beta-pos-4 current_pos) (leg_down-4 current_beta))
(connect (beta-pos-5 current_pos) (leg_down-5 current_beta))
(connect (beta-pos-6 current_pos) (leg_down-6 current_beta))

(connect (up_leg_trigger-1 new_beta) ((suppress (beta-pos-1 signal_pos))))
(connect (up_leg_trigger-2 new_beta) ((suppress (beta-pos-2 signal_pos))))
(connect (up_leg_trigger-3 new_beta) ((suppress (beta-pos-3 signal_pos))))
(connect (up_leg_trigger-4 new_beta) ((suppress (beta-pos-4 signal_pos))))
(connect (up_leg_trigger-5 new_beta) ((suppress (beta-pos-5 signal_pos))))
(connect (up_leg_trigger-6 new_beta) ((suppress (beta-pos-6 signal_pos))))

(connect (walk t1) (up_leg_trigger-1 trig))
(connect (walk t2) (up_leg_trigger-2 trig))
(connect (walk t3) (up_leg_trigger-3 trig))
(connect (walk t4) (up_leg_trigger-4 trig))
(connect (walk t5) (up_leg_trigger-5 trig))
(connect (walk t6) (up_leg_trigger-6 trig))
```

Appendix C Required Motor Torque

C.1 Robot Weight

The weight of the robot was calculated by weighing all the separate parts (see table C.1). The weight of the motor mounts and joints was estimated using blocks of aluminum of roughly the same size.

C.2 Motor Torque

The torque available at the joints was calculated using the manufacturers listed motor torque, and a gear reduction of 30:1 with a 100 percent gear efficiency (see table C.2).

C.3 Required Torque

The torque required was determined as the amount of torque required at the Beta 1 joint for two legs to lift the entire weight of the robot at a chassis height of 12 inches (see figure C.1). This was done by applying the weight of the robot at the center of the chassis, and summing the moments about the Beta 1 joint. At a chassis height of 12 inches, the torque required at a Beta 1 joint = 23ft-lb.

Part	Quantity	Weight/each(lbs.)	Total Weight(lbs.)
Motors	18	1.6	28.8
Shafts	18	.166	3.0
Gears	6	1.1(per leg)	6.6
Screws	6	.27(per leg)	1.6
Joints	18	$.28/rod * .35/yoke$	11.3
Tubing	16 ft	.28	4.5
Mounts	6	1(per leg)	6.0
Total			61.8

Table C.1: Part weights and total weight of robot

motor speed	185 rpm
torque	$220oz - in = 1.145ft - lb$
reduction	30:1
final torque	34.38 ft-lb at 6.17 rpm

Table C.2: Available motor torque

$$\sum M_{\beta_1} = 30.9lbs \times 8.94in \times \frac{1ft}{12in} = 23.02ft - lbs$$

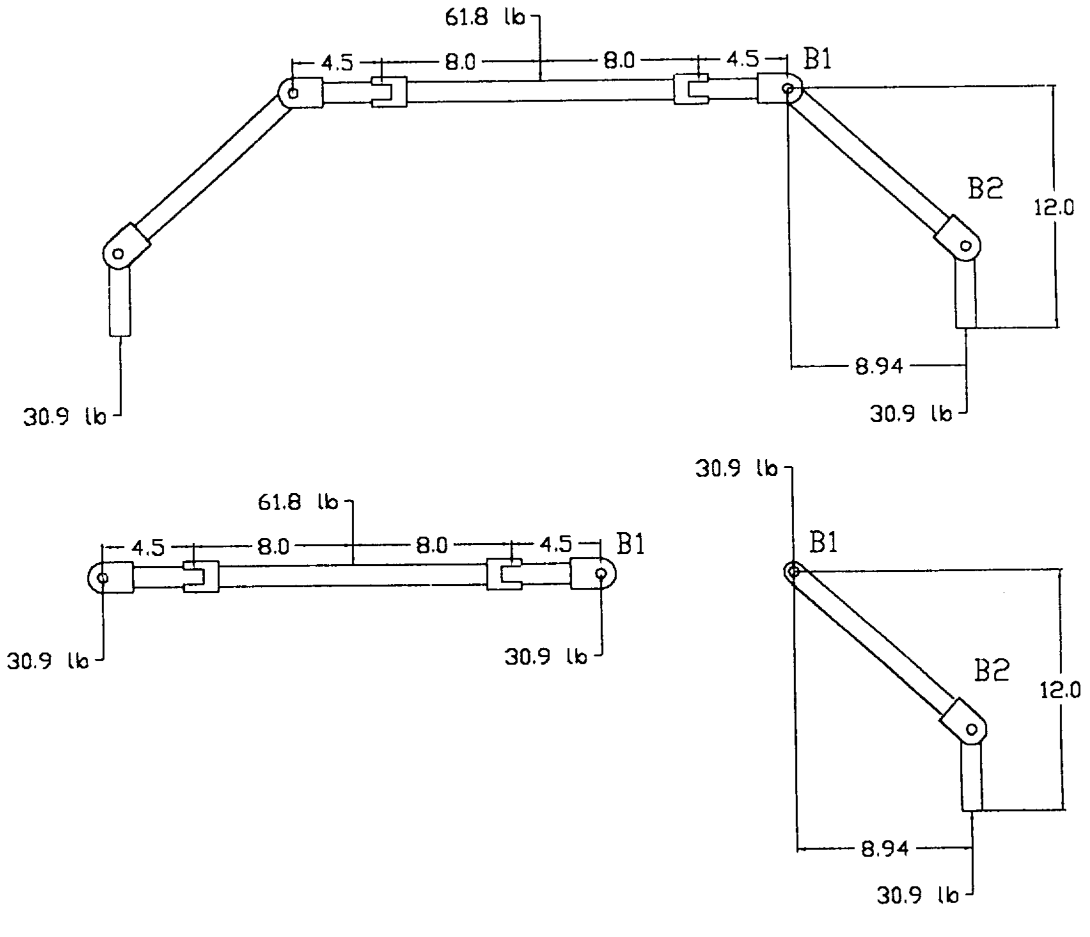


Figure C.1: Moments of Inertia about $Beta_1$

Appendix D Joint Stress

The calculations in this analysis were made using the procedure for stress analysis as described in [Faires 1965] using 6061 T6 aluminum. For ductile materials, a safety factor of six is used.

D.1 Aluminum 6061 T6

The allowable stresses in compression and tension are found by dividing the ultimate stress (see table D.1) by the safety factor. The allowable shear stress is found by dividing the ultimate shear stress by the safety factor.

$$S_C = 7500psi$$

$$S_T = 7500psi$$

$$S_S = 5000psi$$

D.2 Yoke

For the yoke there were five areas that were considered to have high stress concentrations that might weaken or cause the part to fail.

D.2.1 Shearing the Yoke Flange

With the weight of the robot pushing down on one side of the yoke and the rod pushing up on the other side, there was a risk of shearing off the flange of the yoke as seen in Figure D.1.

A calculation was made using the allowable shear stress, S_S, cross sectional area, $A = m \times d$, and maximum allowable force, F.

Ultimate Stress	$S_U = 45000psi$
Ultimate Shear Stress	$S_{US} = 30000psi$
Safety Factor	$N = 6$

Table D.1: Aluminum stress figures

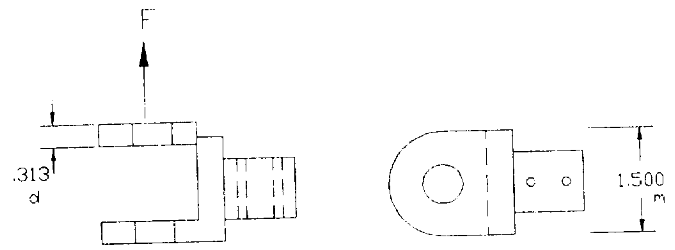


Figure D.1: Shearing of the Yoke Flange

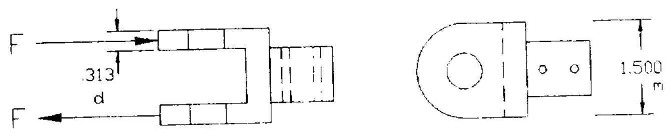


Figure D.2: Compression and Tension in Yoke Flange

$$S_S > 3F/2A$$

Solving for the force gives:

$$F < 2S_SA/3 = 2S_Sm \times d/3 = 2 \times 5000 \times 1.5 \times .3125$$

$$F < 1562lbs.$$

These calculations show that it would require a force many times greater than the weight of the robot to cause any shearing in the part in the vertical or horizontal direction.

D.2.2 Compression and Tension in Yoke Flange

The compressive and tensile forces were determined in the top and bottom of the yoke and are shown in Figure D.2. Tensile forces could possibly pull the flange away and separate it from the rest of the yoke. The compressive forces would act in the opposite direction but would mainly cause deformation in the part.

Using the allowable compressive stress, S_C, and cross sectional area, $A = m * d$, and solving for the allowable force gives:

$$S_C > F/A$$

$$< S_CA = S_Cm \times d = 7500 \times 1.5 \times .3125$$

$$F < 3515.6lbs.$$

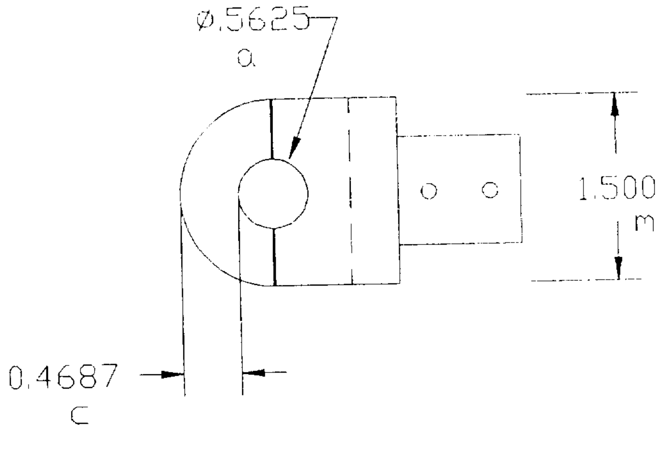


Figure D.3: Shearing Across the Hole

Likewise, using the allowable tensile stress, S_T, and cross sectional area, A, and solving for the allowable force gives:

$$S_T > F/A$$

$$F < S_T A = S_T m \times d = 7500 \times 1.5 \times .3125$$

$$F < 3515.6 lbs.$$

Again, the calculations show that it would take a force of many times the weight of the robot to cause either of these two circumstances to occur.

D.2.3 Stresses for Hole in Yoke

The hole in the flange of the yoke required the final three steps in analyzing the stresses in the yoke. At the hole, the steel shaft would cause stresses in each case. There would be tensile forces across the hole that could split the flange in two pieces. The shaft could shear out of the flange from the across the hole or in front of the hole as shown in Figure D.3 and Figure D.4. Also, the shaft would cause compressive stress on the back of the hole that could deform the part as seen in Figure D.5.
Each of the allowable forces in these cases are determined using the allowable stresses and areas.

Shearing Across Hole

$$S_T > F/2A = F/2b(m-a)$$

$$F < 2S_T b(m-a) = 2 \times 7500 \times .3125 \times (1.5 - .5625)$$

$$F < 4394 lbs.$$

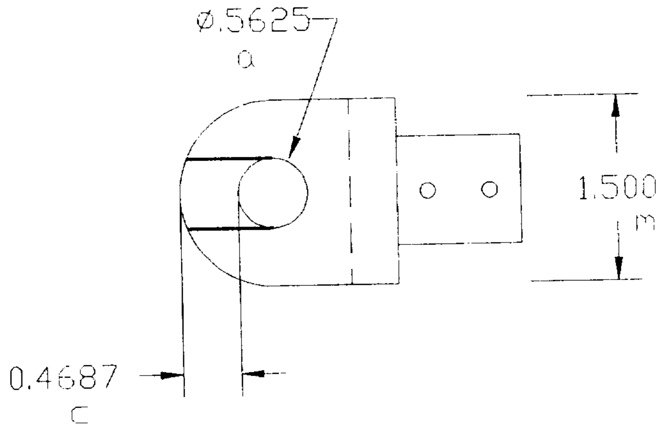


Figure D.4: Shearing Out the Front of the Hole

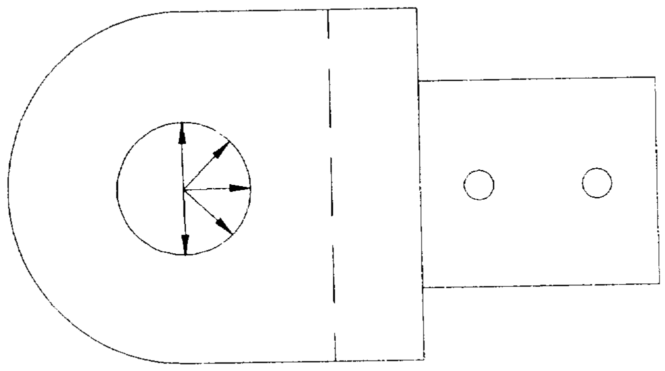


Figure D.5: Compressive Stresses at the Back of the Hole

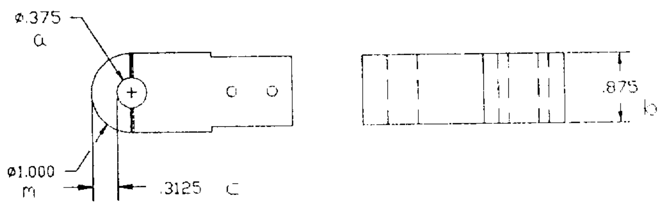


Figure D.6: Shear Across the Hole in the Rod

Shearing Out the Front of the Hole

$$S_S > F/2A = F/2bc$$

$$F < 2S_S bc = 2 \times 5000 \times 3125 \times .4688$$

$$F < 1465 lbs.$$

Compression on the Back of the Hole

$$S_C > F/2A = F/2ab$$

$$F < 2S_C ab = 2 \times 7500 \times .5625 \times .3125$$

$$F < 2636 lbs.$$

All of these allowable forces are far above what would be encountered in the joints.

D.3 Rod

The stress analysis for the rod was very similar to that of the yoke. The steel shaft will cause the same tensile, shearing, and compressive stresses as it did in the yoke. The maximum allowable forces can be found in the same manner as the yoke.

The shaft could cause shear across the hole in the rod as shown in Figure D.6.

The allowable force was then calculated using the allowable stress and cross sectional area.

$$S_T > F/2A = F/2b(m-a)$$

$$F < 2S_T b(m-a) = 2 \times 7500 \times .875 \times (1 - .375)$$

$$F < 8203 lbs.$$

The shaft could also cause shearing in front of the hole in the rod as shown in Figure D.7.

$$S_S > F/2A = F/2bc$$

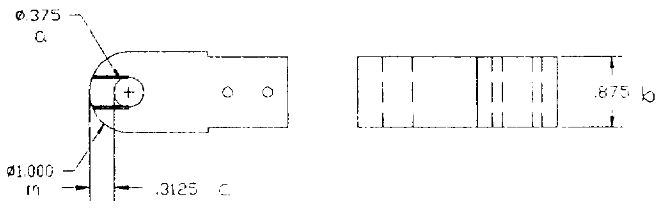


Figure D.7: Shear Out the Front of the Hole in the Rod

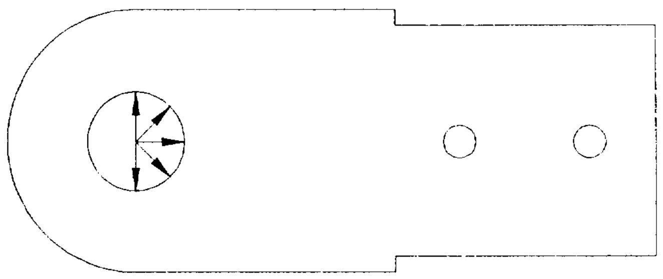


Figure D.8: Compressive Stresses at the Back of the Hole in the Rod

$$F < 2S_S bc = 2 \times 5000 \times .875 \times .3125$$

$$F < 2734lbs.$$

The shaft could also cause compressive stresses at the back of the hole in the rod as shown in Figure D.9.

$$S_C > F/2A = F/2ab$$

$$F < 2S_C ab = 2 \times 7500 \times .375 \times .875$$

$$F < 4922lbs.$$

All of these allowable forces greatly exceed those caused by the weight of the robot.

D.4 Steel Shaft

A stress analysis of the steel shaft was made, using values for ultimate stress and ultimate shear stress (see table D.2). A safety factor of six was used also.

The allowable stresses in compression, tension, and shear were calculated as before, by dividing by the safety factor. The moment of inertia of the shaft was also calculated.

$$S_C = 12000psi$$

Ultimate Stress	$S_U = 72000psi$
Ultimate Shear Stress	$S_{US} = 54000psi$
Safety Factor	$N = 6$

Table D.2: Shaft, C1022 Steel

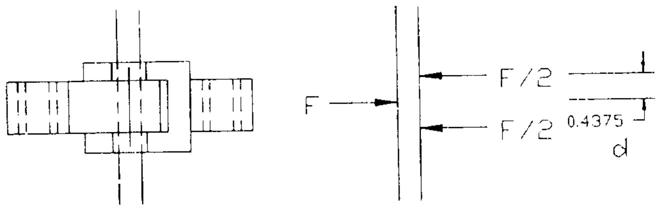


Figure D.9: Stresses in the Steel Shaft

$$S_T = 12000psi$$

$$S_S = 9000psi$$

$$I = PIa^4/64 = .00491$$

D.4.1 Bending of the Shaft

There was a chance that the shaft would bend, as seen in Figure D.9. The force required to bend the shaft while in the joint was calculated. Since the tolerances between the yoke and rod were so small, there would be little chance that the shaft would actually bend in this way.

$$M_{MAX} = Fd/2$$

$$S_T > M_{MAX}y/I = Fdy/2I$$

$$F < 2S_TI/dy = 2 \times 9000 \times .00491/(.4375 \times .2813)$$

$$F < 718lbs.$$

This force is much greater than any that would occur at the robot joint.

D.4.2 Shear with Bending

Another way the shaft could fail would be under the influence of shearing. Since bending was a consideration, it could be assumed that the shaft would bend some amount before it sheared. This was taken into consideration in the analysis of the stresses.

$$S_S > (4/3)(F/2A)$$

$$F < 3S_S A/2 = 3 \times 9000 \times 3.14 \times .5625^2/8$$

$$F < 3355 lbs.$$

Again, it is not likely that any force of this magnitude would be encountered in the robot joints.

Appendix E Complete Parts List

Part Number	Qnt	Description	Supplier
	8	MACHINE SCREW SIZE HAND TAP	MC MASTER CARR
	4	SURFACE-TREATED JOBBERS DRILL	MC MASTER CARR
	21	QUAD AMPLIFIERS, 14 DIP	DIGI-KEY CORP.
L6203	8	MOTOR DRIVER I.C.	MICROCONTROL LAB.
	6	HCTL-2016 DECODER CHIPS	ARROW ELECTRONICS
S6-120	1	SHAFT	BERG INC.
NRB-44	2	ROLLER BEARINGS	BERG INC.
DW116	3	DOUBLE THREADED BRONZE WORMGEAR	CHICAGO GEAR WORKS
DW24H	3	WORMS	CHICAGO GEAR WORKS
SA-120	1	SHAFT	BERG INC.
NRB-65	8	BEARINGS	BERG INC.
3MP28A-20	1	PULLY	BERG INC.
3MF5A-20	1	PULLY	BERG INC.
3CCF-80-E	1	TIMMING BELT	BERG INC.
11-6E L6203	20	MOTOR DRIVERS	HOSFELT ELECTRONIC
	2	PK5 270 OHM 1/4 WIRE	RADIO SHACK
	1	BBD WIE KIT	RADIO SHACK
	1	EXP-300 SOCKET	RADIO SHACK
	1	PCB STANDOFF	RADIO SHACK
	1	DPDT MINI CTR O TOGGLE SWITCH	RADIO SHACK
	3	DPDT MOM 6a F.L TOGGLE SWITCH	RADIO SHACK
	3	HCTL-2000 DECODER CHIPS	ARROW ELECTRONICS
L6202	6	DRIVER CHIPS	ARROW ELECTRONICS
1CB96	2	PC BOARD	RADIO SHACK
	2	LUG	STANION WHOLESALE
	2	PLASTIC TAPE	STANION WHOLESALE
	26	FUSE	STANION WHOLESALE
	1	FUSE BLOCK	STANION WHOLESALE
		MISC. CPVC PARTS WITH BOLTS AND SCREWS	WATERS TRUE VALUE

Part Number	Qnt	Description	Supplier
	5	DELRIN STRIP, 4" W X .101" THICK	MC MASTER CARR
	5	DELRIN STRIP, 4" W X .020" THICK	MC MASTER CARR
	5	DELRIN STRIP, 4" W X .040" THICK	MC MASTER CARR
	100	SOCKET ALLOY STEEL CUP POINT SET SCREW	MC MASTER CARR
	100	18-8 STAINLESS STEEL DOWEL PIN, 1/8" DIA.	MC MASTER CARR
	2	MACHINE SCREWS	MC MASTER CARR
	100	FLAT HEAD PHILLIPS 18-8 SS MACHINE SCREWS	MC MASTER CARR
	10	POLYPROPYLENE BARBED TUBING CONNECTORS	MC MASTER CARR
	36	ONE-PIECE ZINCAL DIE-CAST SHAFT COLLAR	MC MASTER CARR
	1	TAP CARDED, TAP FOR DRILL BIT	MIDWEST APPLIANCE
	1	1" Sq. x .062"WL x 21'1" 6063-T52 AL TUBING	TRIDENT COMPANY
GMX-6MP013A	18	DC MOTORS, MATSUSHITA	SERVO SYSTEMS CO.
W-123A	18	BRONZE WORM GEAR	CHICAGO GEAR WORKS
W-16H	18	STEEL WORMS	CHICAGO GEAR WORKS
S6-120	5	3/8" DIAMETER STAINLESS STEEL SHAFTS	BERG INC.
NRB-65	30	3/8" BORE ROLLER BEARINGS	BERG INC.
	60	15000PF 50V DISC CAPACITORS	DIGI-KEY CORP.
	20	.22UF 63V 20% MONOLITH CERM CAPACITORS	DIGI-KEY CORP.
	20	RESISTOR .51 OHM 5W 5% WIREWOUND	DIGI-KEY CORP.
	20	RESISTOR 5.6 OHM 2W 5% METAL OXIDE	DIGI-KEY CORP.
	40	1 AMP 50 PIV SILICON RECTIFIER	DIGI-KEY CORP.
	6	POWER COOLER 2.0" GOLD W/PIN	DIGI-KEY CORP.
84A1A-B28-J15	8	POTENTIAMETERS	NEWARK ELECTRONICS
	8	RT ANGLE SINGLE M HEADER TIN 36 POS	DIGI-KEY CORP.
	40	4 CIRCUIT TERMINAL HOUSING, .100	DIGI-KEY CORP.
	100	3 CIRCUIT TERMINAL HOUSING, .100	DIGI-KEY CORP.
	100	BRASS/PRE-TIN PIN	DIGI-KEY CORP.
	100	BRASS/PRE-TIN SOCKET	DIGI-KEY CORP.
	200	CRIMP TEMINAL, .100	DIGI-KEY CORP.
	6	TRI STATE OCTAL LINE DRIVER	DIGI-KEY CORP.
	6	(N) TRI STATE D TRI STATE LATCH	DIGI-KEY CORP.

Part Number	Qnt	Description	Supplier
	6	(N) OCTAL D TRI STATE FLIP FLOP SWITCH	DIGI-KEY CORP.
	1	30-AWG BLUE 50FT ROLL WIRE	DIGI-KEY CORP.
	1	30-AWG YELLOW 50FT ROLL WIRE	DIGI-KEY CORP.
	1	30-AWG WHITE 50FT ROLL WIRE	DIGI-KEY CORP.
	1	30-AWG RED 50FT ROLL WIRE	DIGI-KEY CORP.
	60	2 CIRCUIT PLUG	DIGI-KEY CORP.
	60	2 CIRCUIT CAPACITOR	DIGI-KEY CORP.
	25	11-9D SUBMINI MICRO SWITCHES	HOSFELT ELECTRONIC
	100	FLAT HEAD PHILIPS SCREWS	MC MASTER CARR
	100	FLAT HEAD PHILLIPS MACHINE SCREWS	MC MASTER CARR
	200	PAN HEAD SLOTTED MACHINE SCREWS	MC MASTER CARR
	3	MACHINE SCREWS SIZE, HAND TAP	MC MASTER CARR
	36	SPDT 5AMP RELAY 5 VDC UNSEALED	DIGI-KEY CORP.
	20	ROCKERSW DPDT MOM/OFF/MOM/BLK, SWITCHES	DIGI-KEY CORP.
	1	ALUM CONSOLE: 8"x6"x2 3/4"	DIGI-KEY CORP.
	1	BLK-STRANDED-HOOKED WIRE 18 AWG	DIGI-KEY CORP.
	1	RED STRANDED-HOOKED WIRE 18 AWG	DIGI-KEY CORP.
	2	CABLE 40 COND 10' MULTI RIBBON	DIGI-KEY CORP.
	6	ALUMINIUM SQUARE	METAL BY THE FOOT
	6	1.5 ALMINIUM SQUARE	METAL BY THE FOOT
	1	2.5 ALUMINIUM SQUARE	METAL BY THE FOOT
	1	2 ALUMINIUM SQUARE	METAL BY THE FOOT
	5	.5 x 6 ALUMINIUM FLAT	METAL BY THE FOOT