**Worcester Polytechnic Institute**
**Digital WPI**

Major Qualifying Projects (All Years)

Major Qualifying Projects

April 2008

# Two Gait Walking Mobile Robot with Teleoperation Over a Wireless Network

Kevin J. Harrington
*Worcester Polytechnic Institute*

William Travis Kurzmack
*Worcester Polytechnic Institute*

Follow this and additional works at: https://digitalcommons.wpi.edu/mqp-all

# Two Gait Walking Mobile Robot
# With Teleoperation Over a Wireless Network

A Major Qualifying Project
Submitted to the faculty
of the
Worcester Polytechnic Institute
In Partial Fulfillment of the requirements for the
Degree of Bachelor of Science

Submitted By:
William Kurzmack

_____

And

Kevin Harrington

_____

Advisor:

Yiming (Kevin) Rong

_____

Due: 4/24/2008

# <u>Abstract</u>

The robot provides a platform for individuals with limited mobility to navigate  urban terrain. The system should have minimal impact on its environment, be very stable during movement, and be easily integrate wireless network systems. Minimal impact on the environment, was achieved using a hexapedal robot using a triangular paired-leg design. The hexapod was given two separate gaits, one for normal walking and one for negotiating  stairs. This allowed the center of gravity to remain low while normally walking. The  tele-operation was conducted over a wireless internet connection using a central  dial home   server. The robot was to designed to each criteria successfully. During the project, future additions were considered, making the robot a good platform for future projects.

# Table of Contents

# Illustration Index

# Introduction

  More then 11,000 cases of spinal cordial injuries (SCI) occur every year (Spinal, 2006). Of these cases, approximately 9,000 of the patients are permanently paralyzed in some way. In the most extreme cases patients can become paraplegic or even quadriplegic. It has been estimated that the cost of caring for a individual suffering from quadriplegia at age 25, over their entire life span, will amount to $2,924,513.  Much of the cost is due to other people having to perform many of the everyday tasks for patients.

  However, monetary and statistical data doesn't give a full picture of what of what is going on. Most paraplegic and quadriplegics rely on some sort of assisted living.  This creates a sense of dependence, causing patients  to feel frustrated and even helpless. It also means that  patients are very often dependent on other peoples schedules for simple things, like a glass of water.

   The goal of this project is to design and build a teleoperated assisted living robot, able to navigate urban terrain for those with limited mobility. The robot will need to be able to address stairs, as well as, other sharp elevation changes. It should be also be capable of dealing with sloped surfaces. The robot must also be small enough to fit through a doorway without precise orientation. The gait chosen should be reversible and create a high level of static stability through the entire cycle, so the robot doesn't tip. The overall impact of the robot on its surrounding environment when walking should be minimal. To control the robot a simple and intuitive web applet will be developed to and communicate via a wireless network.

  The desired result of the project is a functioning prototype. the prototype should be able to accomplish the aforementioned goals, and be adaptable for future additions and projects. Upon completion of the project new options mobility will be available for those suffering for SCI. the project will also help the field of robotics by improving options in wireless control and making a cheap conversion for DC motors to servos readily available, and updating an older gait design to be more tolerant in elevation changes.

# __Background__

There were three major considerations before entering the design phase, they were: the current state of products available to SCI patients, a set of standardize testing metrics for robots, and previous designs. Since, there are currently no marketed teleoperated robots for SCI patients, research on what was available was conducted. Similarly, very few metrics for comparing robots were available. The most complete and recognized was based on the Robocup Rescue competition. Finally, a review of the prior art was completed to learn more about possible design paths.

## *Current State*

Most of those patients with serious cases of SCI are wheelchair dependent. It is estimated that only 50% of the Earths land is accessible to wheel or tracked vehicles (Raibert, 1986). This percentage is even lower for wheelchairs. In cities, were much of the ground is paved, there are still problematic areas, such as curbs or stairs that hinder wheelchair users. If it is taken in to consideration that the wheelchair is powered by its user then the slope of a surface can become more problematic. Currently, government standards require that all new buildings have handicap ramps with a slope of no more then 1:12 or about 5 degrees (Access-Board. 2007). However, older structures and other areas of a city may not comply with these standards.

Even with these many limitations great strides have been made in recent years to address these problems, increasing both mobility and independence for those living with SCV. One example, in the case of paraplegics, is that iBOT wheel chair. The chair is able move up stairs by balancing one set of wheel atop of another, giving it the height it needs to go up a step. This ability, to balance on one set of

wheels, has also had an added advantage in that it brings the patient up to eye level with a person that is standing. This helps the patient to feel that they are not being talked down to when talking with others. Unfortunately, the iBOT runs around $29,000 in cost (About the iBOT,2007). This makes it too expensive for most paraplegics, in many cases quadriplegics can't use it at all. It also has the same problems as most wheelchairs in that it difficult to maneuver through tight places like doorways, hallways, or even through a class room with desks. It also has difficult with some rough terrain, and can damage carpet over time.

One of the simplest ways to increase maneuverability in tight areas is to reduce the size of the platform. This is difficult in the case of a wheelchair since it must be large enough to carry an patient. It would also be difficult to reduce the the impact that the wheels will have on carpet or increase a chairs ability to handle rough terrain. It seems like there will always be limitations for wheelchair users. However, this is assuming that the wheelchair user must be physically present to perform the task. In many cases this isn't true, the user could simple direct someone or something to perform the task for them, then bring them the results.

With this in mind, it seems that a different approach could be taken to improve how to tasks are accomplished for those with limited mobility. The different approach would use robot to retrieve things for its user. This would allow for a smaller size, since it wouldn't have an occupant, making it more maneuverable in a home or urban environment. A legged design could be utilized to reduce wear on surfaces. It would also be advantages for dealing with incline surfaces and stairs, which traditionally present a major problem for wheelchair users. It will also have the advantage of being able to step over

items that have been left on the floor, something a wheel or treaded robot would find difficult to accomplish.

Another consideration is an autonomous design, such as the robots used for the DARPA Urban challenge. However, since it is the fundamental goal to provide the operator with a since of presence in the environment the robot navigates, the design would counteract the goal. There is also the advantage of reduced computations and higher adaptability if a human operator is involved. This is one of the reason telepresence, were an entity controls a host remotely, has become very appealing in recent years.

## *Robocup Rescue Robot Competition*

Robocup consists of several robotics competitions on the international level (Robocup. 2007). It is perhaps best known for its robots that play soccer. However, one its more serious competitions is the Rescue Robot Competition. The competition is designed to simulate urban hazards found in a partially or completely collapsed building. These standards are similar to those used by the National Institute of Standards and Technology (NIST), who helps sponsor the competition, for testing actual robots used in rescue situations. There are several main goals for the Rescue Robot course, they are: negotiation of obstacles, mapping, and detection of signs and other objectives.

## *Robocup Obstacles*

There are many obstacles in the Rescue Robot Competition. obstacles range from ramps to crawling through duct work. An overview of the obstacles that will be used for this project is given below. For specific building specifications please see appendix A.

1. Examples: Step Fields



*Illustration 1: Red Step Field (Robocup Rescue Robot 2007) League*



*Illustration 2: Red Step Field (Robocup Rescue Robot League 2007)*

Step fields provide a practical challenge to robots. Step fields can come in many different arrangements. The simulated uneven terrain attempts to force the robot to high center by placing low elevation blocks near high elevation ones. The blocks are not actually attached to the palates that they are arranged in causing any side load, applied by the robot, to move them. This requires not only a good mechanical design, but excellent feedback from the robot, regarding surroundings, to the operator.



*Illustration 3: PVC Step (Robocup Rescue Robot League 2007)*

2. Examples: Stairs and PVC StepThe PVC step and stairs both provide interesting and unique challenges. In the case of the PVC, the edge is able to freely rotate, making it difficult for robots to simple drive over the surface. This makes it very difficult for tank



*Illustration 4: Stairs (Robocup Rescue Robot League 2007)*

tread style robots to negotiate this particular obstacle.

The stairs, on the other hand, provides another interesting challenge. Since the stairs are slotted, and have no back board, they're somewhat difficult to detect with a simple bump or touch sensor. Also, without a back board, a robot is

forced to pull itself onto the stairs.

    3.   Examples: Sloped Floor and Ramps

The sloped floor is not usually considered a major obstacle from a navigation stand point, though it can through off detection sensors. However, for an legged design, a sloped surface may require some design consideration. so that the robot does not simply slide down a surface or fail to walk up it. This same consideration needs to be applied to ramps as well.

## A Brief History of Walking Mobile Robotics

The initial use of walking mechanisms were as toys. The first documented walking mechanism was in the 1870's. It was designed by P.L. Chebyshev, a Russian mathematician, who wanted to mimic natural walking(Santos, Pablo G., et al. 1985, pg 3). Later designs would be for military application and would seek to create vehicles with legs. This would prove too difficult, without the aid of computer automation, and most early attempts resulted in abandonment. One of the most notable of these attempts was in the 1940's by A.C. Hutchinson. His design used individual feedback loops for each leg, and made use of a rolling thigh joint. However, the project was never attempted on at full scale.

Real attempts at walking transports and robots were not made again until the 1960's. This was due to several advancements. Most notably the classification of animal gaits by Hildebrand and the development of formulas for gaits by McGhee and his contemporaries (Santos, Pablo G., et al. 1985, pg 15). Such advancements eventually lead to the GE walking truck, which was in many ways a success as a walking machines; However, the design proved too demanding of its driver, requiring back legs to be controlled by the drivers feet and the front by his hands, to be of practical use.

Another interesting development was that of the iron mule, which used eight rather then the traditionally favored four leg design. legs were pair closed to gather on each side making a set of four gangs . A system of cams and sprockets were used to link the legs to gather on each side so that only two motor needed to be used. Unfortunately, this design proved to be limited in its adaptability to terrain. It also proved very difficult to turn. If anything, the iron mule and the GE walking truck are some of the last examples of completely mechanical solutions to the problem of walking (Todd, DJ 1985. pg 17).

The 70's and 80's saw the further development of legged robot design. The reduction of the size of electronics allowed for more adaptable designs. One of the earliest attempts was by R. McGhee. Having seen the GE walking truck, McGhee decided that the design could be adapted to be controlled by a stepping cycle generator controlled by a computer. This lead to the construction of a robot known as the Phony Pony. the Design had 2 DOF legs, which could be made to move forward, backwards, or lock in place. Signals were interlocked to stop all the legs from moving at once. However, lateral stability was problematic, and was resolved by using wide feet on the chassis (Todd, DJ. 1985, pg 24).

Another early example of computer controlled robots was the Ohio State University hexapod. Another similar design was the PV-II. This eventually, lead to more practical robots like Dante II, which was used in the early 90's to investigate Volcanoes(Votaw, Brooks).

The final major break through came in the late 80's, with the development of linear actuated systems that could obtain variable gaits. First among these designs was by Raibert, who used several linear actuators and a hip gimbal design to control gaits. This basic idea is used in most modern

walking machines such as Big Dog, and Timberjack[1].

Another interesting development, in late 1989, was the Genghis. Genghis was simple 2 DOF autonomous walking robot. However, it was able to demonstrate relatively complex behavior with only four microprocessors. It also developed a very simplistic gait which was nicknamed the Genghis gait (Retired Robots, 1996).



*Illustration 5: Genghis (Retired Robots, 1996)*

In recent years there has been many developments in walking robots. One of the most sophisticated is Big Dog, produced by Boston Dynamic. It hydraulic driven legs are controlled by on board computes. However, the truly unique development is its ability to stabilize its self after an impact force is applied.  It also tries to actively recycle energy from stepping, and is able to carry a 120 lb load. Another unique aspect of Big



*Illustration 6: Big Dog with packs (Boston Dynamic 2007)*

Dog's design is its articulated legs. This leg design is very similar to those found in animals, such as dogs.

---

1    Although the Timberjack is no longer manufactured a video of the Timberjack can be found at
     http://www.youtube.com/watch?v=CD2V8GFqk_Y

# __Methodology__

## *Design Requirements*

       Listed below are a set of the key design criteria for the project. The two major elements driving these criteria were the Robocup Rescue metrics and future platform adaptability. While the Robocup Rescue metrics clearly help to define the mechanical design, future adaptability insured that the end user or future project teams could easily change out electrical components or add additional mechanical mechanism to the frame.

Design Specifications:

1. Mechanical Requirements
    1. Gait
        1. Frame is statically stable during entire gait cycle
        2. Gait is reversible
        3. Gait is tolerant to sharp elevation changes (7in for stair climbing)
        4. Gait is tolerant to gradual elevation changes
    2. General
        1. Robot must be able to clear 27in wide door way in any orientation
        2. Robot must be able to carry a 10lb payload
        3. Robot must minimize impact on surfaces it walks on
            1. Carpet
            2. Tile
            3. Wood
            4. Linoleum
2. Electrical
    1. Robot will run on battery power
    2. Robot should have a 2 hour life span per charge
    3. When inactive power consumption should be minimized
3. Programming and Networking
    1. Web based user interface
        1. Controls should be simple and intuitive to use
        2. Images from camera should be update in real time or as near as possible

2.  Robot should be able to switch from one wireless node to another with no difficulty
3.  If a wireless connection is lost the robot should remain immobile until the connection is reestablished

## *Mechanical*

## Sensors

One of the key features of the robot was its ability to be controlled remotely. To accomplish this the robot required some sort of sensors to send back information to the operator. The primary sensor chosen for this task was a web camera. The major reasons being: cost, usability, and functionality. As web cameras have become more popular their price has dropped, so that a reasonable one can be purchase for somewhere between thirty to sixty dollars. Since the camera is already designed for use over the web it is easy to connect to  this kind of system. Also, the low resolution of the camera makes it easier to send over a wireless network. It also has advantages over other sensors due to it large range. Finally, only one camera is needed to control the robot. This is due to the cameras ability to show obstacle range as well as height. The camera will require a pivot, so that the operator can see were they are going.

While ultrasonic sensors were considered they proved to be too expensive, usually between thirty and three hundred dollars for one sensor. Ultrasonic sensors also proved to have a very short range depending on their resolution, generally between eighteen centimeters and four meters. Finally, ultrasonic can only see in small bands making them ideal for range, but difficult to see obstacles out side the band. This can be resolved by using multiple sensors, but the amount of formating to make the data readable and useful to the operator would be difficult and slow down updating.

In addition to a camera, limit switches have been added to the legs. This allows the robot to check for static stability when the robot is moving to a new position. If three limit switches are pressed, then the robot knows it legs are in valid locations. If the tree limit switches are not then the robot will stop moving forward and wait for the operator to change directions.

## Number of Legs

There are a two major trade offs when it comes to leg selection. The fewer legs a systems has the less complex the systems will be, however; the fewer legs a system has the smaller its stability footprint. While four legs provides adequate support, if only one leg is moved at a time, the movement of the robot will be slow. It would also have difficulty lifting itself onto the stairs with only one free leg.

Eight legs would also prove problematic, due to the added control mechanisms required to control the extra legs, and the increased risk of failure. However, eight legs would provide greater stability when trying to climb stairs.

These leaves the optimal choice of six legs which will allow a statically stable base when climbing stairs but also allow a statically stable system to be waiting for the chassis when it changes positions.

## Gaits

While there are numerous gaits, many of which are still undocumented, a insect like gait was chosen for the final design. The gait does not match a specific type of insect, but was based off the general motion of insect gaits. This is due to the tendency of insect gaits to always remain statically stable when walking. This is accomplished by only lifting up three legs at a time. This set up is ideal for

having a stable base when climbing stairs. However, due to increased stability it will not be as fast as other gaits, such as a quadruped dynamic gait.

## Degrees of Freedom

Each leg element required three degrees of freedom in order to move forward and walk up stairs. While some robots like Genghis were able to generate very simple gaits with only two DOF, they proved unable to negotiated sharp changes in elevation, however; Genghis was able to move fairly quickly. To resolve, a two DOF end effector(Red) was created for normal motion. To turn, another DOF was added about the vertical axis of the leg(Green). Since independent turning of each leg is unnecessary, rotation about the axis was chained together. Finally a redundant vertical motion(Yellow) was included in the leg for a
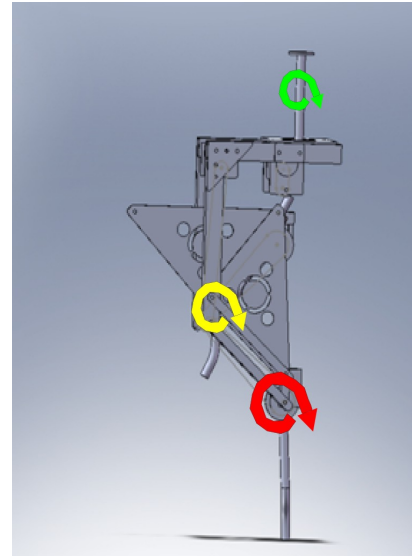


*Illustration 7: Locations of DOF*

separate high stepping mode for stairs and other obstacles. If this redundant DOF hadn't been added the average cycle time of a normal step would have been very long and result in a great deal of wasted vertical motion.

## Motion Generation

In many traditional robotic applications motion generators are a major consideration, and very dependent on application. This project was no exception. Traditionally the main form of motion generation has been servo motors, stepper, motors, pneumatics, and hydraulics. While all have certain advantages they also have disadvantages. Once again prior played a major role in helping to determine

the final decision. In the end it was decided to adapt normal DC motors to function as servos.

Hydraulic Systems:

Hydraulics excel at high load applications, that require precise repeatability. Unfortunately the system notoriously dirty, making it unusable in domestic environments. It also is very difficult to design hydraulics to handle multiple positions; traditionally the system only actuates on and off. The overall weight of the system also make it difficult to put on a light weight mobile system.

Pneumatics:

Pneumatics proved to be problematic for several reasons. Pneumatics are generally inefficient at energy transfer, which makes them less ideal for a battery powered platform. Pneumatics also have problems holding position under heavy loads, due to compression of the gas. Similar to hydraulics, pneumatics traditionally have two position that they can hold: fully open and fully closed. This is problematic if a stair step is a different height. Pneumatics also tend to be heavy when used on a limited scale. In almost all of the prior art, pneumatics systems tended to limit the gait type to that of insects styles or require very difficult control loops.

The Stepper Motor:

The stepper motor uses a series of electromagnetic teeth or steps to align the motor with a given position. There are usually several sets of steps that are slightly miss aligned with one another so that the motor will move forward when they are activated. While steppers use a simple open loop control system, making them easy to control, they require full continuous power. Since the robot will not be

tethered, continual draining of the batteries will reduce operation time considerable. Stepper also

increase dramatically in price as the resolution of the motor increases. This is due to the increased

manufacturing cost from adding more electromagnetic steps to the system.

The Servo motor:

The final choice was servo motors, which seemed to be the best choice for a small scale project

like this one. While they require constant input as to were their location is, servo don  t require full

power all of the time like steppers. It is also easier to attain a higher resolution with servos at a lower

cost. The only major problem is that servos are usually sold as small scale hobby motors, making large

scale motors expensive and difficult to come by. This option would most likely require a DC motor to

be converted to a servo by hand

## CAD Modeling
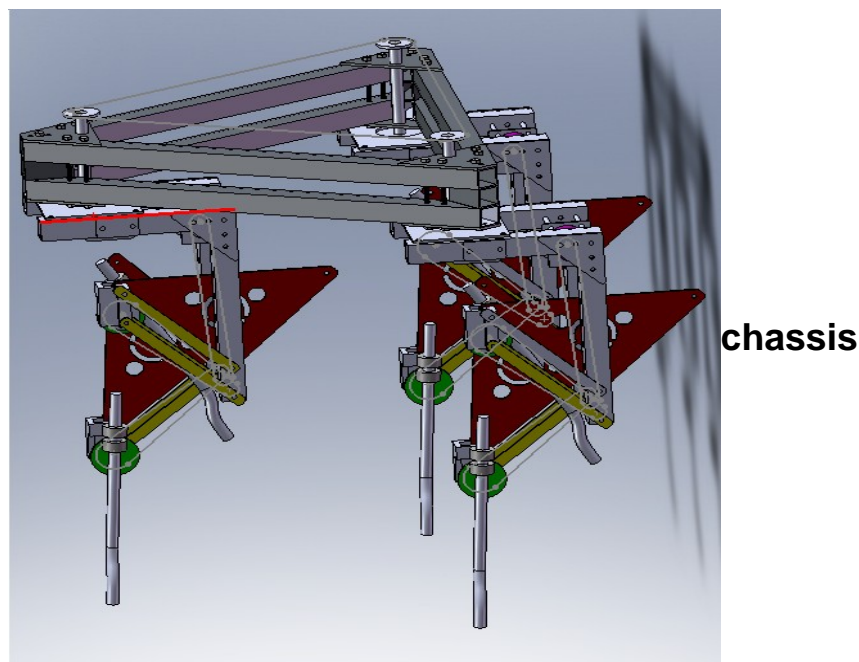
**chassis**

*Illustration 8: Full Mechanical CAD Model of Robot*

To model the robot, systems were broken into the chassis and leg. the chassis was the most obvious of the three, and was completed first. As many of the mates that could be were added to the bolts and nuts in the assembly model. Densities were assigned to each part. Originally, gussets plates were run along the side of the chassis to allow bolts to hold a tube for wire running from the leg. This was later disregarded, for top and



*Illustration 9: Orignal Chassis Design*

bottom gussets, which were easier to manufacture and more stable. the double decker design was also dropped in the final prototype.
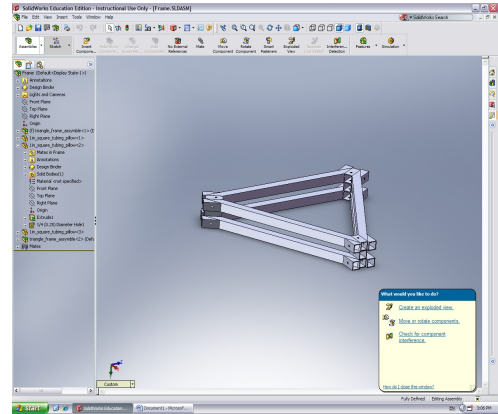
## Legs

The leg was completed in two parts one as a stationary assembly section and a linkage section. The stationary assembly consisted of the leg frame and mounting brackets for the motors. Whenever possible, parts were mating at holes, to remain as accurate as possible. All parts were set to be static.

Each of the linkage parts needed to move in some way. To accommodated that freedom was given about the axis or direction needed for the walking motion. Parts consisted of the thrust bearings, chain, four bars, and most he revolute joint.

### *Electrical*

One of the main design decisions made for the electrical system was the inclusion of a complete computer on board the robot. Traditionally embedded systems have been used on robots, due to more efficient power consumption and the advantages of dedicated systems. For this project, because the

application of the robot was very general, it was felt that a more adaptable system was required. It also affords the advantage of being able to handle additional sub systems, as well as, different types of data inputs and outputs. Another major advantage was that wireless hardware and connectivity had already been resolved.

Power:

The power supply for this the system was relatively simplistic. Lithium polymer cells were used for the battery. Lithium Polymer was chosen due to it increasing availability and light weight. Output from the battery was controlled using a buck converter. the converter can handle from 12 to 36 volt inputs, and output to a 5 volt, 3 amp rail, as well as, 12 volt, 6 amp rail. While a linear regulator could have been used to perform this task it would only have achieved 40% efficiency. With the buck converter 96% efficiency was achieve in the conversion.

Motor Interfaces:

The motor controller has two variations, one is an eight port hobby servo interface controller and the other is a PWM, PD, servo controller. Both take commands from the rs232 interface embedded into the PIC16f916. They both use the serial interrupt module with the same Interrupt Service Routine to determine what the data is that is streaming in over the serial port. The addressing of each module is performed in the high bit of the 8 byte stream, leaving 7 bits of each byte for data information. Each byte is associated with one channel of control. The controller reads the entire 8 byte stream and if it gets an addressing bit it is not expecting then it drops the entire frame and resets waiting for a stream addressed to it. Once it gets a properly addressed stream it updates the register controlling the motor on

all channels in the case of the hobby servo interface controller or the single motor position in the case

of the servo controller. For the servo controller the position input is compared to the current position

given by the A/D converter module and a potentiometer attached to the output shaft of the motor. After

the comparison, the motors PWM is adjusted to move the motor to the desired position.

## *Programming*

The main goal of the code was to send information from the user to the robot, so it would move

in the appropriate direction. Visual data also needed to be sent to the operator in real time, to allow the

operator to make appropriate decisions. Below is a diagram showing how each piece of code is used in

the process of communicating with the robot. A full print out of the code can be found in appendix D.

All code has been released under the general public license (GPL).

**Servos and Camera**

**Camera** —— RS232 ——

**Robot's Computer**
**Python Script Serializes Code**

**Python Main Script**

**Hobby Servo**

**DC/Servo Conversion**

Control Inputs

**Server**
**Python Script**

Data Dump for Picture

Control Inputs

**Operator's Computer**
**Java Applet**

Data Dump for Picture
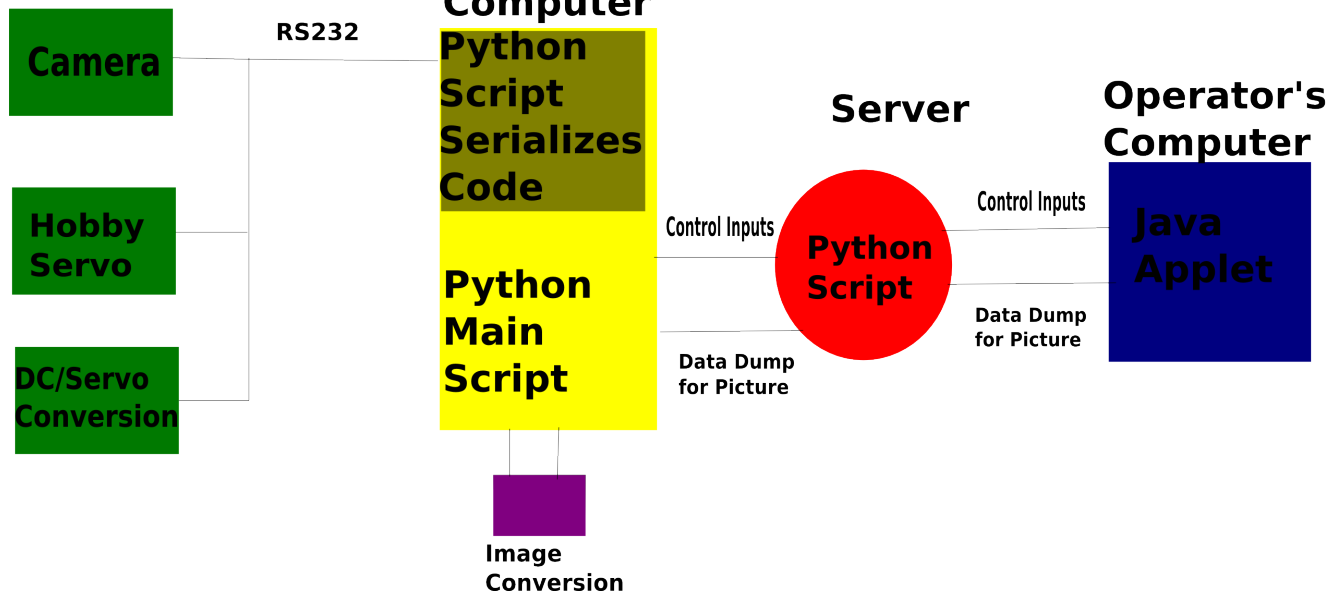
**Image Conversion**

*Illustration 10: Program Layout*

Three major languages were used to for the system. A Java applet was used for the operator

interface program. The information was then sent to a python server program, which directed it to the a

second python script. This information was then serialized by a third python script and sent the servo.

At the servos, the code was handled by PIC code and the new servo position was set.  The resulting

system was the due the design criteria, and consideration for the physical components.

PIC:

Due to the 16F916 PIC used for this project, PIC was the only real language option when

coding for the servo controllers. Potentially, C could have been used, but would not have been as precise

as programming directly in PIC. Initially, the code was done procedurally, requiring constant   polling

by the PIC to see if a signal had been sent. This resulted in a high load on the chip. To eliminate this

problem, the code was rewritten with interrupts, which waited for the a signal rather then polling for

one.

Python:

The Python segment consisted of three different pieces of code. The first piece of code

serialized the input from the user. The second piece, which was the main code, sent control inputs to the

first python script. The program  also converted images from the camera to bitmap using the open cv

library, before sending a data dump of the image to the sever. The final python script sent inputs stored

at the server from the user to the robot. It also sent images stored at the server to the user by performing

another data dump to the Java applet. No buffering was used, so as soon as the image or control input

was received it was sent out.

Originally, C was used for the much of the code. The result was over a hundred lines of code

that proved difficult to edit and expand. To resolve this, python was adopted for the entire

communication process, which simplified communication and code length. This is large part due the

python being a higher level language, with extensive modules available for coding. Networking was

simplified using the Py Twisted module, which reduce the server side code to fifteen lines.

Java:

      The Java program acted as the user interface, providing real time images from the robot, and

allow the operator to direct the camera and the robot as needed. The main reason for using Java was due

to the high level of functionality inherent in the language. Java is also a widely accepted language

making it installation on Windows or Linux operating systems simple. Flash could also have been used

in this application, no major advantage would have be received.

      To minimize an operators transition time to a new system a standard video gaming interface

was used. Forward backwards and left and right were controlled by  w ,  s ,  a , and  d . The camera s

pan and tilt mechanisms, were controlled by using the mouse to point and click on  the current camera

image. Based on were the image was clicked the camera would shift in that direction. This was later

refined, so that the mouse only needed to be moved for camera motion to occur. As throughout all the

code, buffering was not used, to insure a real time image. While the decision sacrifices control over the

frame rate as the wireless signal increases and decreases, it insures that the operator is able to discern

what is in front of them at all times.

# Results
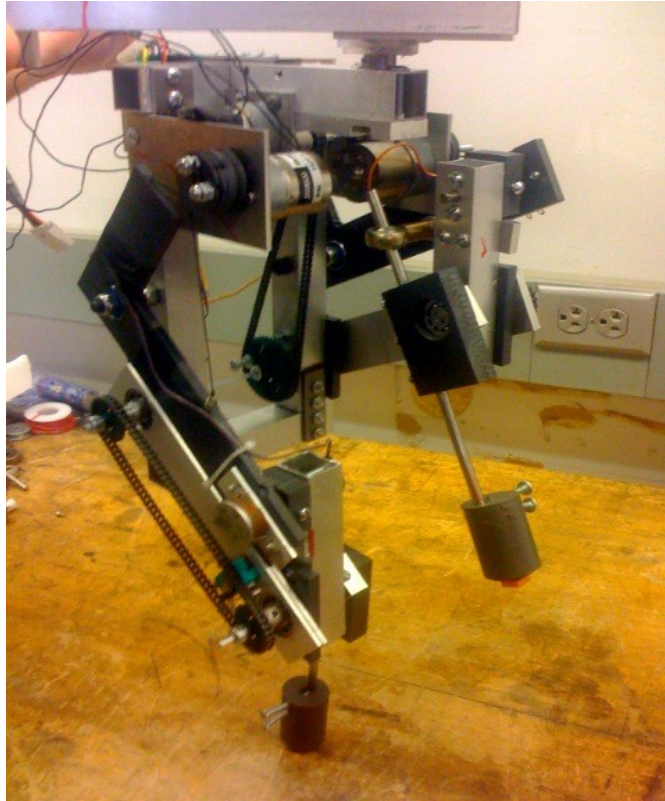


*Illustration 11: Leg Functioning Under no Load*

Although the model was able to be built and was completed, several problems occurred that are still being resolved at this time. The original design called for a chain and sprocket system using VEX parts. This proved advantageous for rapid assemble, but due to a inaccurate tension rating provided by VEX, the chain was unable to handle the loads placed on it. To resolve this problem, 3.75 mm pitch steel chain was adopted. This allowed the sprockets from the VEX kit to be reused. Unfortunately, due to the uncommon size of the chain, it proved cheaper to replace all the sprocket and chain with ANSI standard #25 chain and sprocket. Since the tension rating is for 140lb, no problems are anticipated when using the chain upon its arrival.

Since the robot is not currently operational, no quantitative data of its mechanical performance could could be taken. However, several qualitative observations can be made about the system. Also, the electrical and code for the robot is functioning correctly, and quantitative result are available for those parts.


*Illustration 12: Example of VEX Chain*

With no load the robot was able to complete its normal gaits without difficulty. A complete cycle, under no load, takes approximately 1 second with a horizontal displacement of 3inchs. If the robot was able to walk, it would be able to move at about 5 yards per minute.

After further refinement, the static stability in the design achieves its goal of a highly stable platform. It is most likely tip in high step mode, which was anticipated. The gait is reversibles, as required.

The environmental impact the robot when walking could not be tested. However, due the small surface area in contact with the ground, the nature of the gait, and the soft plastic material chosen for the foot pads, impact on carpet or other home floor surfaces should be minimal over time.

The camera is able to send images at 15 frames per second over the a local network. The signal was also send out California from WPI over commodity internet. A frame rate of 2 frames per second was achieved, which is acceptable for operation. A lag time is currently based on the update time between images as a result of a real time update design.

A connection time of 1 second was achieve for the robot to the network. This has been

consistently achieved. To improve communication time the python scripts are design to take advantage of a dual core processor system.

## **Future Work**

While the project focused on resolving several key elements of an assisted living robot for those suffering from SCI, more still needs to be done. Currently, the robot is able to walk to a desired destination; however, it only allows the operator to view not interact with its surroundings.

There is currently no audio system on the robot. Without this feature, the operator is unable to communicate or receive audio information about the area its walking through. To properly address the problem, a two way communication system needs to be set up. Since a computer, rather than an embedded system, has been used for the I/O, open source VOIP programs, such as Ekiga, could be adapted to fill this need. Otherwise, a real time compression program for raw audio will have to be designed and implemented. A small amount of buffering could be accomplished at the server to reduce delays in speech. However, the audio will need to be synced with the visual data, which is currently unbuffered. If the visual and audio data is not synced it may be difficult for the operator to discern who is speaking. Buffering may also cause problem with the operator knowing when to speak.

One very noticeable mechanism missing from the robot is a manipulator. While a manipulator was well outside of the scope of this project, it will be essential in addressing the over all problem of an assisted living robot. Specifically, in order to be able to navigate an urban environment completely, so

sort of door opening mechanism will need to be designed. The manipulator will also need to perform

tasks that the operator may want complete. Since tasks will vary considerably, a minimum six DOF arm

will most likely be required. Due to a large DOF number, joint control will most likely difficult for the

operator to use. To improve control a world or end effector coordinate system will need to be designed

and implemented for the end user. A second camera and other sensors may be needed to help the

operator control and direct the manipulator.

## Conclusions

Many of the goals set for the project were

achieved. A stable wireless connection with server to

user communication was designed and implemented.

Images feedback from a webcam on the robot was

able to be sent to the operator. The DC motors used

on the project were successfully converted to servos.

The legs of the robot were able to complete their gait

cycle under no load.
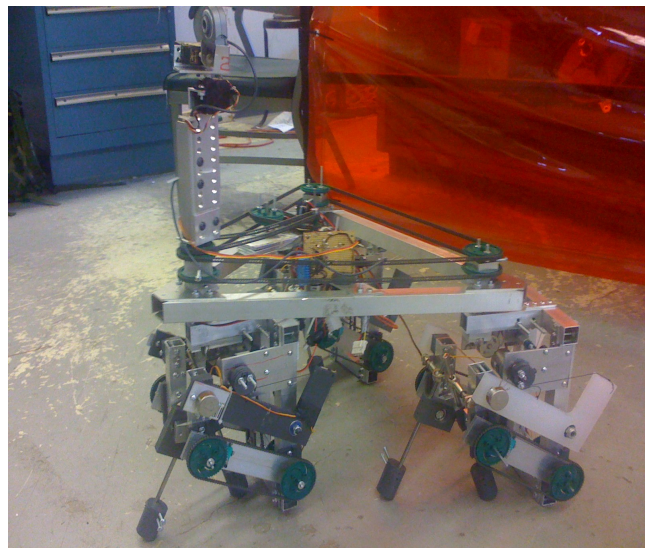


*Illustration 13: Current Prototype With
Functioning Webcam*

Unfortunately, due to problems with the chain used for prototyping, the robot was not able to

function under its own weight. While this problem is currently being resolved, it will not be fixed in

time for the papers deadline. For this reason no quantitative testing was able to be conducted.

Qualitatively, it appears the robot meet all its design criteria, and would have performed well in testing

with the Robocup obstacles

The project will continue till the end of the term, at which time the robot should be fully functional. Future work will be continued as an ISP by a project member. An MQP may be setup to accomplish some of the suggestions made the future work section.

# <u>Works Cited</u>

- About the iBOT. (2007). Independence Technology. http://www.ibotnow.com/about-ibot.html November, 23, 2007

- Access-Board. (2007). http://www.acess-board.gov/ufas/ufas-html/ufas.htm#4.8 November, 21, 2007

- Boston Dynamic. (2005). http://www.bostondynamics.com/content/sec.php?section=BigDog. November, 21, 2007

- Braunl, Thomas. (2006). Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems. Berlin, Germany: Springer

- Coiffet, Philippe.(1983). Robot Technology: Interaction with the Environment (volume 2). London, England: Prentice-Hall International Inc.

- Everett, H.R. (1995). Sensors for Mobile Robots: Theory and Application. Wellesley, Massachusetts: A K Peters, Ltd.

- Florczyk, Stefan. (2005). Robot Vision: Video-based Indoor Exploration with Autonomous and Mobile Robots. Germany: Wiley-VCH Verlag GmbH & Co. KgaA

- Heiserman, David L. (1976). Build Your own Working Robot. Blue Ridge Summit, PA: Tab Books

- Kauffmann, L'hote, et al. (1983). Robot Technology: Robot Components and Systems (Volume 4). Englewood, NJ: Prentice-Hall, Inc.

- Nehmzow, Ulrich. (2006). Scientific methods in Mobile Robotics: Quantitative Analysis of Agent Behavior. Germany; Springer

- Nehmzow, Ulrich. (2000). Mobile Robotics: a Practical Introduction. Berlin, Germany; Springer

- Norton Robert L. (2004). Design of Machinery: An Introduction to the Synthesis and Analysis of Mechanisms and Machines (3rd edition). Boston, Massachusetts: McGraw Hill

- Raibert, M.H. (1986).Legged Robots That Balance. Boston, Mass: MIT Press

- Retired Robots, (1996). http://groups.csail.mit.edu/lbr/genghis/ November, 10, 2007

- Robocup Rescue Robot League. Sponsored by NIST (2007). http://wiki.cc.gatech.edu/robocup/index.php/RoboCupRescue_Robot_League November 10, 2007

- Spinal Cord Injury Information Network. (2006). University of Alabama. http://www.spinalcord.uab.edu/show.asp?durki=21446 November, 21, 2007

- Santos, Pablo G., et al. (2006) Quadrupedal Locomotion: An Introduction to the control of Four-legged Robots. Germany: Springer

- Todd, D.J. (1985) Walking Machines: An Introduction to Legged Robots. New York, New York: Chapman and Hall

- Vertut, Jean, et al. (1986). Robot Technology: Teleoperations and Robotics: Evolution and Development (Volume 3A). Englewood, NJ: Prentice-Hall, Inc.

- Vertut, Jean, et al. (1986). Robot Technology: Teleoperations and Robotics: Applications and Technology (Volume 3B). Englewood, NJ: Prentice-Hall, Inc.

- Votaw, Brook. Telerobotic Applications. http://www1.pacific.edu/eng/research/cvrg/members/bvotaw/ November , 27,2007

# **Appendices**

## *Appendix A*

Site contain Robocup building requirements for field elements:

http://www.rescuesystem.org/robocuprescue/

## *Appendix B*

code

Java Applet:

import java.applet.*;

import java.awt.*;

```java
import java.awt.event.*;
//import java.awt.image.MemoryImageSource;
import java.io.*;
import java.net.*;
//import java.io.BufferedReader;
//import java.io.InputStreamReader;
//import java.io.OutputStreamWriter;
//import java.io.PrintWriter;
//import java.net.ServerSocket;
import java.net.Socket;
//import java.util.Date;
//import java.util.Vector;
import javax.imageio.*;
import java.security.AccessControlException;

public class ImageNetListener extends Applet {
        private int touchX;
        private int touchY;
        private int touchXOut;// = (touchX/2)+128;
        private int touchYOut;// = (touchY/2)+128
        static final long serialVersionUID = 1;
        public Socket pyServer;
        public DataOutputStream os;

        public File file;
        public Image img = (Image)null;
        //Global DataInputStream in;
        public void init() {
                System.out.println(os);
                System.out.println(pyServer);
```

```java
            System.out.println("init pyServer and os");
            touchX=getSize().width/2;
            touchY=getSize().height/2;
            addMouseListener(new TouchpadMouse());
            try{
            URL hostURL = new URL(getParameter("URL"));
                    try {
                        pyServer = new Socket(hostURL.getHost() , 8886);
                        os = new DataOutputStream(pyServer.getOutputStream());
                            System.out.println(os);
                            System.out.println(pyServer);
                            System.out.println("established pyServer and os");
                    } catch (Exception x) {
                        System.err.println("Couldn't get I/O for the connection to");
                    }

            }catch(MalformedURLException j){
                j.printStackTrace();
            }

            new Server(8885);

    }
    public void paint(Graphics g) {
            g.drawImage(img, 0,0, getSize().width, getSize().height, Color.blue, this);
            g.setColor(Color.black);
            g.fillRect(touchX, touchY, 2, 2);
            g.drawString("test", touchX, touchY);
    }
private class TouchpadMouse implements MouseListener {
```

```java
String xout;

String yout;

TouchpadMouse (){

}

public void sendByets(String e) {


        try {

                 os.writeBytes(e);

                // os.close();

                } catch (Exception a) {

                        System.err.println("IOException: in send bytes " );


                }

}//send bytes


public void mousePressed(MouseEvent e) { }

public void mouseReleased(MouseEvent e) { }

public void mouseEntered(MouseEvent e) { }

public void mouseExited(MouseEvent e) { }

public void mouseClicked(MouseEvent e) {

        float width = getSize().width;

        float height = getSize().height;

        float getx = e.getX();

        float gety = e.getY();

        float divx = getx/width;

        float divy = gety/height;

        float xtouch = divx*256;

        float ytouch = divy*256;

        //int  xtouch2 = (int)((e.getX()/getSize().width)*128.0);
```

```java
                touchX = (int)xtouch;

                touchY = (int)ytouch;

                //System.out.println(xtouch2);
                touchXOut = (touchX/2)+128;
                touchYOut = (touchY/2)+128;
                xout=new Integer(touchXOut).toString();
                yout=new Integer(touchYOut).toString();
                sendByets("0,"+xout+",1,"+yout);
                touchX = e.getX();
                touchY = e.getY();
                repaint();
            }
        }
    private class Server extends Thread {
            public final static int DEFAULT_PORT = 8885;
            protected int port;
            protected ServerSocket listen_socket;

            // Exit with an error message, when an exception occurs.
            public void fail(Exception e, String msg) {
                System.err.println(msg + ": " +  e);
                System.out.println("Server Thread generic");
                System.exit(1);

            }

            // Create a ServerSocket to listen for connections on;  start the thread.
            public Server(int port) {
```

```java
            new Connection(pyServer);


        }


        // The body of the server thread.  Loop forever, listening for and
        // accepting connections from clients.  For each connection,
        // create a Connection object to handle communication through the
        // new Socket.
        public void run() {


        }//run
    }//server


private class Connection extends Thread {
        // protected Socket client;
        protected DataInputStream in;
        // protected PrintStream out;


        // Initialize the streams and start the thread
        public Connection(Socket client_socket) {


          try {
            in = new DataInputStream(pyServer.getInputStream());
          }
          catch (IOException e) {
            try {
                    pyServer.close();
            }catch (IOException e2) {
                    ;
                    }
```

```java
        System.err.println("Exception while getting socket streams: " + e);

        return;

    }

    this.start();

}
// Provide the service.
// Read a line, reverse it, send it back.
public void run() {

    try {
        InputStream is = new BufferedInputStream(in);
                    while (true){
                        System.out.println("waiting for input stream");
                        try {

                                os.writeBytes("`");
                                //os.close();

                                } catch (Exception a) {

                                        System.err.println("IOException: in send bytes " );


                            }
                        System.out.println("image request sent");
                        img = ImageIO.read(is);


                        System.out.println("setting image");
                        repaint();


                        Thread.sleep(20);


                        System.out.println("repaint");
                    }
```

```java
                }catch (AccessControlException s){

                    System.out.println(s);

                            //g.drawString("fail"+s, 8, 8);

                    }catch (Exception e){

                        System.out.println(e);

                        //g.drawString("FAIL", 8, 8);

                    }finally {

                        try {

                                pyServer.close();

                        } catch (IOException e2) {

                                System.out.println(e2);

                        }//catch

                    }//finally

            }//run

        }//connection


}//NetListenetn
```

Python Server:

```python
#!/usr/bin/python


import os
import sys


from twisted.internet import reactor
from twisted.internet.protocol import Protocol, Factory


send = Factory()
send.robot = None
get  = Factory()
```

```python
get.client =None
#push = Factory()
class Sender(Protocol):
    def connectionMade(self):
        self.factory.robot = self
            print "connection Robot"
    def connectionLost(self, reason):
        self.factory.robot = None


    def dataReceived(self, data):
      # print "image sent"
        try:
                    get.client.transport.write(data)
            except:
                    print "applet not connected"


class Receiver(Protocol):
    def connectionMade(self):
        self.factory.client = self
            print "connection Applet"
    def connectionLost(self, reason):
        self.factory.client = None


    def dataReceived(self, data):
            print "data sent"
            try:
                    send.robot.transport.write(data)
            except:
                    print "robot is not connected"
```

```python
if __name__ == '__main__':
        send = Factory()
        send.protocol = Sender
        #push.protocol = Sender
        get = Factory()
        get.protocol = Receiver
        reactor.listenTCP(8887, send)
        reactor.listenTCP(8886, get)
        reactor.run()
```

Python Program:
```python
#!/usr/bin/python

import serial, sys, os, time

from twisted.internet.protocol import Protocol, ReconnectingClientFactory, Factory
from twisted.internet import reactor

from twisted.protocols import basic

from threading import Thread

import pygame
import Image
from pygame.locals import *
import sys
import opencv
from opencv import highgui
import cStringIO
dataArray = [0 for i in range(0,16)]
```

```python
ser = serial.Serial('/dev/ttyS0', baudrate=19200, bytesize=8, parity='N', stopbits=1, timeout=1)


get = Factory()
lc = Factory()
get.robot = None
f = Factory()
f.client = None


camera = highgui.cvCreateCameraCapture(0)
fps = 1
pygame.init()


def get_image():
    image = highgui.cvQueryFrame(camera)
    #convert Ipl image to PIL image
    im = opencv.adaptors.Ipl2PIL(image)
    pg_img = pygame.image.frombuffer(im.tostring(), im.size, im.mode)
    pygame.image.save(pg_img, "/media/ramdisk/image.bmp")
    os.system("killall -9 nc")
    #os.system("killall dd")
    os.system("dd if=/media/ramdisk/image.bmp|nc 127.0.0.1 8885 &")
    print "image sent"


class ListenClient (basic.LineReceiver):
    def connectionLost(self, reason):
        dataArray[0:4] = 192, 160, 140, 192


    def connectionMade(self):
        dataArray[0:4] = 192, 192, 255, 192
        self.factory.camBridge.dataOut = self
```

```python
    def sendImage(self, data):

        self.transport.write(data)


### START CLEAN
    def dataReceived(self,data):

        #print data
        offset = 5
        test255 = 255-offset
        test128 = 129-offset
        tmp1 = dataArray[1]
        tmp0 = dataArray[0]
        if data.find('~')==0:
            speakTest=data.strip().strip("~")
            dataArray[15]=1
            os.system("echo "+speakTest+"|espeak")
            dataArray[15]=0

        elif data.find('`')==0:
            print "image requested"
            im = get_image()


        elif data.find(',')>=0:
            print data
            i, v, j, z = data.strip().split(',')
            print i
            print v
```

```python
            print j
            print z
            xVal=int(v)
            yVal=int(z)
            xInv=128
            yInv=128
            for b in range(0,xVal):
                xInv=xInv-1
            for c in range(0,yVal):
                yInv=yInv-1
            xInv+=255
            yInv+=255
            print xInv
            print yInv
            dataArray[0] = xInv
            dataArray[int(j)] = yInv
            print data
        else:
            print data
### END CLEAN


####Protocal handeling
class Receiver(basic.LineReceiver):
    def dataReceived(self, data):
        if self.factory.dataOut:
            self.factory.dataOut.sendImage(data)
        else:
            print "No connection"
```

```python
class ClientFactory(ReconnectingClientFactory):
    def startedConnecting(self, connector):
        print 'Started to connect.'
    def buildProtocol(self, addr):
        print 'Connected.  \nResetting reconnection delay'
        self.resetDelay()
        lc = ListenClient()
        lc.factory = self
        return lc
    def clientConnectionLost(self, connector, reason):
        print 'Lost connection.  Reason:', reason
        ReconnectingClientFactory.clientConnectionLost(self, connector, reason)

    def clientConnectionFailed(self, connector, reason):
        print 'Connection failed. Reason:', reason
        ReconnectingClientFactory.clientConnectionFailed(self, connector, reason)


#Robot serial contol
class SendSerial (Thread):
    def run(self):
        openJaw = True
        while (True):
            s = ""
            if dataArray[15]:
                if openJaw:
                    print 'Passed if openjaw==1'
                    dataArray[2]=160
                    openJaw=0
                    print openJaw
                else:
```

```python
                dataArray[2]=220
                openJaw=1
        else:
            dataArray[2]=220

        for j in range(1,262):
            for i in dataArray:
                s += chr(i)
            ser.write(s)
            ser.flushOutput()
        time.sleep(.005)#smooths out the loss of packets


if __name__ == '__main__':
    SendSerial().start()

    get = Factory()
    get.protocol = Receiver
    get.dataOut = 1

    f = ClientFactory()
    f.protocol = ListenClient

    f.camBridge = get

    reactor.connectTCP("hephaestus.cs.wpi.edu", 8887, f)
    reactor.listenTCP(8885, get)
    reactor.run()
```