5-31-2005

A toy rock climbing robot

Matthew P. Bell
Dartmouth College

A toy rock climbing robot
Honors Thesis

Matt Bell

May 31, 2005

Dartmouth College Computer Science Technical Report
TR2005-542

Abstract

The goal of this thesis was to build a simple toy rock climbing robot, and to explore problems related to grasping, path planning, and robot control. The robot is capable of climbing a wall of pegs either under manual control through a host system and an infrared interface, or on the basis of a set of pre-recorded keyframes. In addition, the robot can climb certain peg configurations using a cyclic gait. The robot climbs in an open-loop mode without sensor feedback. All communications are sent through the IR connection, and the tether to the robot consists only of two power wires.

Contents

List of Figures

1 Introduction

The goal of the climbing robot project is to develop a simple toy robot capable of climbing a wall of pegs. A major focus of this project is to keep the robot as simple as possible in order to make it feasible for the general public to buy an inexpensive kit for building the robot. There are three climbing modes:

1. Manual remote control
2. Autonomous, with pre-recorded keyframes
3. Autonomous, using a simple cyclic gait

The robot is capable of climbing under manual control through a Java interface and autonomously on the basis of a set of pre-recorded keyframe positions. For appropriate wall configurations, a set of cyclic keyframes exists that will make the robot climb the wall with a cyclic gait. When executing a pre-recorded sequence of keyframes, the robot climbs open-loop, with no sensor feedback. The robot receives all of its control communications through an infrared (IR) receiver, and as a result, the robot's tether consists only of two wires for power.

In addition to building a simple toy robot, this thesis explores problems related to grasping under uncertain conditions. Most robot grasping problems involve grasping a small object with a large industrial manipulator; our robot can be thought of as a small manipulator grasping the entire climbing wall. Due to the lack of sensor feedback, the robot is unable to correct its position if it gets off course; it is necessary to plan motions in a way that provides significant stability and repeatability.

Knowledge of grasping under the more constrained conditions of the climbing wall can be extended to more complex problems, including the addition of a third dimension. To simplify analyses of the climbing robot, it is considered to be operating in only two dimensions. Adding a third dimension to this system would require additional servos on the robot to enable it to navigate in this dimension, which would greatly increase the complexity of the analyses. However, a robot capable of 3-dimensional climbing should be capable of climbing any surface with protrusions that can be gripped. Such a robot might also be capable of rotating its arms downward and walking on them, making it suitable for a general purpose mobile robot capable of operating in difficult terrain, such as rubble.

Section 1 discusses related climbing robot projects, and examines possible applications along with the research contributions of the toy climing robot project. Section 2 details the physical construction of the robot, along with design decisions that were made during the robot's development. Section 3 explores the problem of planning a path up the climbing wall on both a high and low level, and also looks at alternate methods for climbing the wall. Section 4 discusses the mathematics of the climbing robot, including the forward and inverse kinematics and stability calculations. Section 5 describes the software that is used to control the robot, including host software and embedded software on the robot. Finally, Section 6 details the ways in which the robot was able to climb.

1.1 Related Work

Our toy robot is not the first climbing robot; it is, however, the simplest, one of the lightest, and the only one capable of climbing without sensor input or significant computation. In this section, we review three other climbing robot projects.

A group at Dartmouth under the supervision of Stephen Linder is currently working on Tenzing [5], a climbing robot initially developed during the CS88/188 course in Spring 2004. Tenzing is considerably more complex and heavier than the toy robot, and it employs several sensing methods to assist in motion planning. It interprets a live video feed to determine where the robot is in relation to the handholds, and it has force feedback sensors in its feet to determine if it has a good grip or not. The general shape of Tenzing is similar to the toy robot, although it is larger. In addition, Tenzing has feet at the end of the outer arm sections that rotate freely and are used to actually stand on handholds. This difference allows Tenzing to stand with its arms at any angle with respect to the handhold, whereas the toy robot must place its arms directly on the handholds at an angle that prevents it from losing its grip. However, this enables us to explore whole-body manipulation. Although the robot as a whole is prehensile, the arms themselves are non-prehensile.

Timothy Bretl of Stanford has developed a path planning algorithm for JPL's LEMUR II robot [1]. This robot is capable of climbing walls with arbitrarily shaped and angled handholds, utilizing movements similar to those that a human climber would use. The design of this robot is such that there is no definite "up" orientation of the robot body. This allows the robot to maneuver much more freely on the wall, as the body can rotate around without affecting the range of motion of the arms. The LEMUR II is similar to Tenzing in that its arms can be at any angle with respect to the handholds, as the end effector on its arms is a small peg wrapped in high-friction rubber. The LEMUR II is a fairly heavy robot, with a mass of 7 kg. It also has some capability of moving in 3-dimensions, as the outer arm joints can pitch away from the climbing wall. This gives the robot some freedom to move over handholds with at least the outer sections of its arms, rather than having to move around them.

Finally, Michigan State University has a pair of climbing robots that use suction to climb on smooth walls and ceilings [8]. The two robots, Flipper and Crawler, use different principles for their movement. Flipper has its suction cups at the ends of a V-shaped pair of arms. As its name implies, it flips end over end as it moves across a surface. The robot's motion is very similar to brachiation (swinging from handhold to handhold). However, the robot does not need to search for a specific handhold; instead, all it needs to do is to contact the surface with the suction cup on the end of the arm. Crawler has a straight frame that is capable of extending to provide forward motion. Crawler is designed to be very small and lightweight, weighing only 450 grams. The general motion of Crawler is similar to a caterpillar's motion, except for the fact that Crawler does not lift its body to move forward. Both Flipper and Crawler are more complex due to their use of suction to maintain handholds, and are only capable of climbing on smooth surfaces.

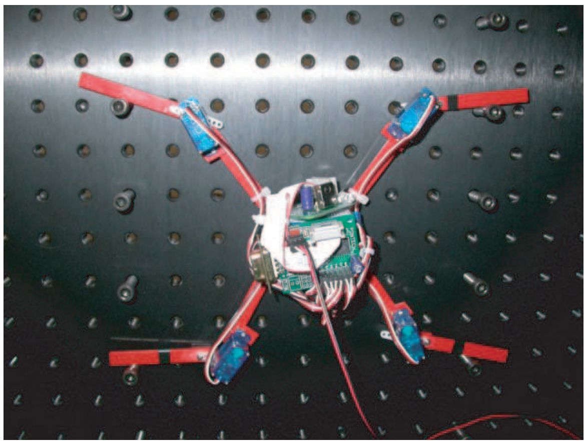


Figure 1: Photograph of the robot

1.2 Applications

The toy climbing robot will be of use to both the general public and to the robotics community. As outlined earlier, the public will benefit by being able to play with a simple robot, which should increase public interest in robotics. There are no plans to market this particular robot at this time, but the simple design allows for such a possibility in the future. One such possibility is to work with Acroname (`http://www.acroname.com`) to develop and market a kit. Acroname has previously worked with MIT and CMU, and has marketed CMU's Palm Pilot Robot Kit.

1.3 Research Contributions

The mimimalist design of the robot poses a number of interesting challenges:

1. Whole-body manipulation
2. Regrasping
3. Open-loop grasping
4. Cyclic gaits
5. Mobile manipulation

As the climbing robot does not have any specific end effector for gripping handholds, it uses whole-body manipulation to grasp the climbing wall. Any part of the robot can be used for gripping a handhold. For example, there are situations where it is advantageous to contact a handhold with an elbow servo or even with the inner arm. Although it can be more difficult to plan for a system utilizing whole-body manipulation, it makes the system much more flexible.

The problem of regrasping can occur in any sort of industrial system in which a part needs to be regrasped from a different orientation. The climbing robot extends the idea of regrapsing more steps into the future than most typical applications. The robot regrasps the wall every time it moves an arm to a new handhold. Thus, over the course of an entire climb, the robot may regrasp the wall more than 30 times.

Open-loop grasping is a significant challenge, which again comes up in many industrial systems. Many solutions such as vibratory positioning systems and special fixtures exist to orient parts without the need for sensing. In all of these cases, the key is to execute a series of motions to place the object to be grasped in a known configuration relative to the manipulator. For a simple example, consider trying to grasp a book from an arbitrary configuration. A method for doing so involves dropping the book into a bin and tilting the bin so that the book always ends up in the same corner of the bin. The climbing robot makes use of a similarly simple method to attain a known configuration when it is using a cyclic gait to climb.

Cyclic gaits are related to open-loop grasping. As just mentioned, it is necessary to force the robot into a known configuration to make it possible to repeat the cycle correctly. If the robot does not have a recentering move, it may progressively move off course, which will eventually lead to the robot falling. One recentering strategy that was developed for the climbing robot involves hooking the elbow joints over handholds.

As the climbing robot can be thought of as both a mobile robot and as a manipulator grasping the wall, it is essentially a mobile manipulator. As such, it explores the fact that mobility and manipulation can be achieved by the same set of controls.

2 Robot Design and Construction

The initial construction of the robot was done by Devin Balkcom, my thesis advisor, over the course of the Christmas break. This initial design was refined by the author during the course of the thesis project. This section reviews the physical components of the robot, and some of the design decisions that were made.

2.1 Physical Description

Parts List

- Climbing Wall:
  - Optical Bench Plate (2x2 ft)
  - 1/4-20 screws
- Robot
  - 8 1x8 flat LEGO pieces
  - 1/2" pieces of additional 1x8 flat LEGO pieces
  - 8 micro servos
  - Glue
  - 1/4", No. 0 screws
  - Pontech SV203C (controller and IR receiver board)
  - Velcro
  - Cable ties
- Control System
  - Host system with Java version 1.5
  - Serial cable
  - Pontech SV203C (controller and IR transmitter board)

The robot body was formed from 4 micro servos that are glued together (Figure 2). The

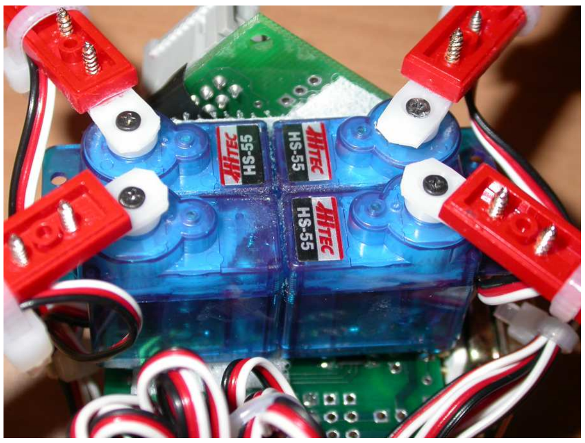


Figure 2: Servos that make up the robot's body

use of the servos as the robot's body made it unnecessary to build any sort of frame for the robot, which simplified the design and reduces the robot's overall weight. The arms are 1x8 flat LEGO pieces, and were attached to the servos with screws and glue. The outer arms are controlled by 4 additional servos attached to the ends of the inner arms. Using a direct drive system, with the elbow servos placed on the arms, resulted in a simpler design as compared to a pulley and belt system, with the servos mounted on the robot's body. The servo wiring was bundled together with standard cable ties. Figure 1 is a photograph of the robot.

The servos are controlled by a Pontech SV203C board, which is attached to the robot with Velcro strips. The SV203C can be programmed using a subset of BASIC known as SVBASIC. This board provides control for 8 servos, and is powered by a 6V power supply; power regulation is built in to the SV203C. A daughter board consisting of an IR transmitter and receiver provides IR functionality. The IR board on the robot only utilizes the IR receiver. The host computer communicates with a second SV203C through a serial connection; this second board also has an attached IR board, of which only the transmitter is utilized.

An optical bench plate and 1/4-20 screws were used to create a climbing wall with handholds for the robot. The optical bench plate was used for the wall because it can be easiy reconfigured by rearranging the screws. Also, because all of the holes are threaded, the screws are very stable, which is advantageous for improving the repeatability of motions. A pegboard of the variety that is used in a toolshed is a possibility for a cheaper, lighter climbing wall, as is a bulletin board with pushpins.

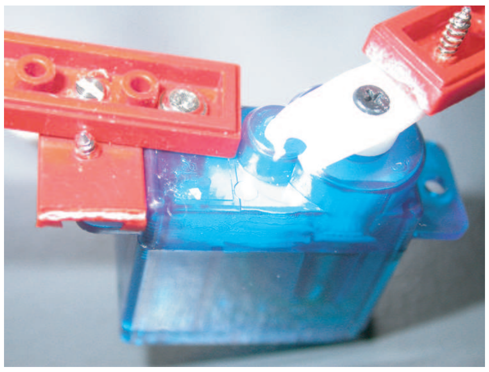


Figure 3: Method of attaching servos to inner arms (the inner arm is on the left)

2.2 Design Decisions

Two main changes were made to the basic design of the robot, involving the servo attachment and the communication method. The servos at the elbow joints were initially only attached to the inner arms by one screw, which allowed the elbow joint to rotate slightly. Additional screws and LEGO pieces were used to attach the elbow servos to the inner arms at two points (Figure 3).

Communication with the robot was initially accomplished through a standard serial connection. The serial connector on the robot end of the cable created some difficulties due to its size and weight. In particular, the robot tended to tilt away from the wall, and the connector was in the way of one of the elbow servos. The serial communication method was replaced with IR communication to remedy this problem.

The robot does not receive any feedback on its progress on the wall because it does not utilize any sensors. Sensors are an additional expense, and would have required additional wiring and might have required a different type of controller board, depending on the number of sen-

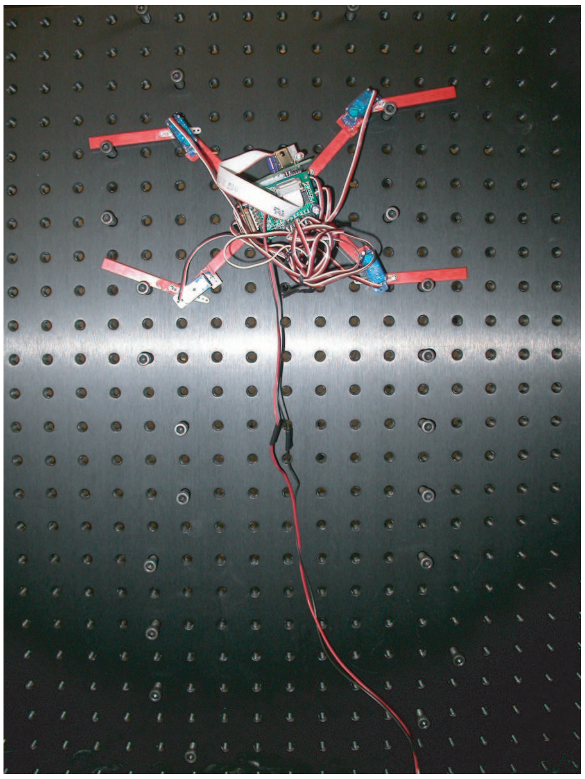


Figure 4: Climbing wall configuration

sors. However, sensors might make it easier to climb the wall. Some possibile sensor configurations are discussed in Section 3.1.

The climbing wall was configured by placing the screws in a ladder-like configuration (Figure 4). This was done in part because of limitations of the robot. The body of the robot is in the same plane as the arms, and as a result, the robot cannot get past any obstacles that the body might run into. For the gait, an even more regular screw configuration was used. It consisted of two vertical lines of evenly spaced screws. Future work includes exploring other wall configurations and climbing methods, such as doing a traverse (sideways motion) across the wall.

3 Path Planning Problem

The basic goal of the path planning algorithm is to get the robot to the top of the wall. Along the way, the robot must avoid obstacles (in other words, it should not run into a peg with the body or with the wrong side of an arm), and it must be in a stable configuration throughout the climb.

The path planning task can be split into high-level and a low-level tasks. The high-level plan determines what overall sequence of handholds should be used to climb the wall. The low-level plan determines joint movements on a handhold-to-handhold basis. The high-level plan may need to at least use approximations of the low-level plan in order to successfully compute a route; however, this may not be necessary for all wall configurations.

For the ladder-like configuration of the wall, a relatively simple high-level plan can be used. Specifically, the robot will move its lower two arms to higher handholds. Next, the robot will shift its body upwards while leaving the arms at the current handholds. Finally, the robot will move its upper two arms to higher handholds, and possibly shift the body again if necessary. This cycle will repeat until the robot reaches the top of the wall.

Due to the fact that the two sides of the ladder may not be perfectly vertical series of handholds, it is necessary to perform low-level path planning for each movement of the robot. There are two main subtasks associated with the low-level planning. First, the planner needs to determine a sequence of joint movements that will move an arm from the current handhold to a new handhold without colliding with any other handholds on the way. This is accomplished through the use of inverse kinematics calculations, which are detailed in Section 4.2. Basically, inverse kinematics maps a desired (x, y) location for a specified point on the arm to the joint angles that will cause the arm to be at the specified location. Using inverse kinematics, it is possible to calculate the joint angles that will result in the arm resting on the new handhold. To get from the current handhold to the new handhold, the end of the arm should follow a curve that keeps the arm from colliding with the handhold (Figure 5). Inverse kinematics can map this $(x(t), y(t))$ curve to a $(\theta_1(t), \theta_2(t))$ curve to determine exactly how to control the arm.

The second subtask of the low-level planner involves calculating the stability of a given grasp of

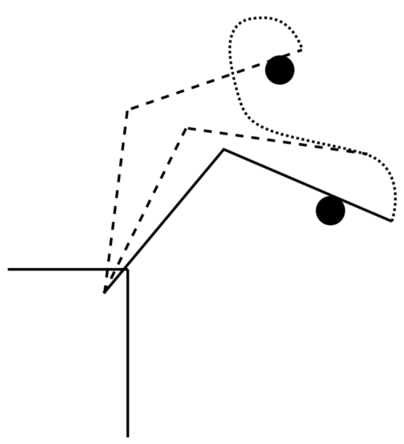


Figure 5: Example of curve used for low-level path planning

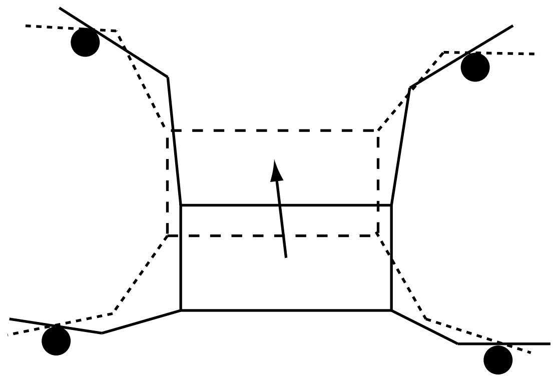


Figure 8: Example of robot shifting body while maintaining handholds

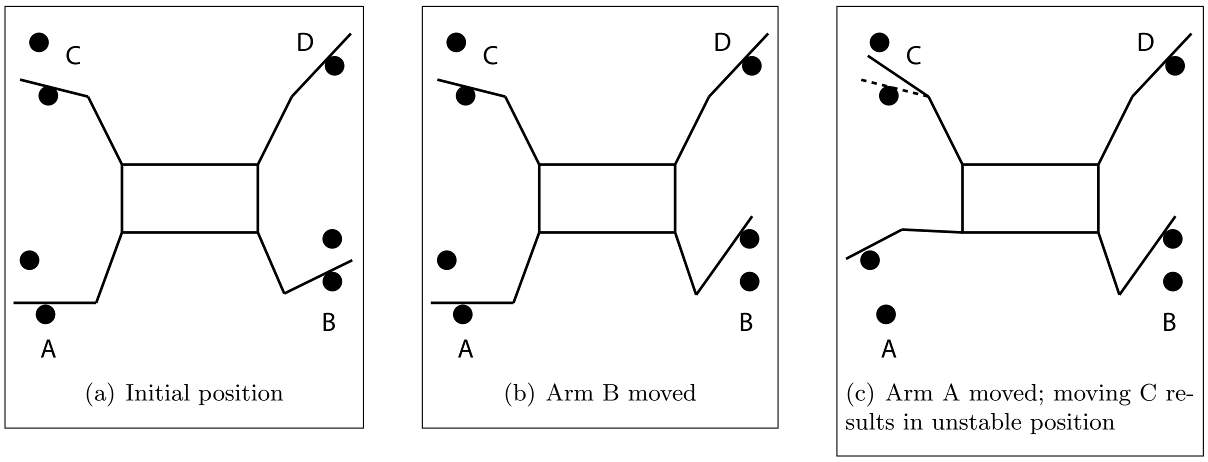
the wall. Methods for computing stability will be discussed in Section 4.3. It is not enough for the robot to be stable in a given grasp. The low-level planner must also take into account portions of the high-level plan. There are several questions that the planner must take into consideration:

1. Is the current configuration (with all 4 arms contacting handholds) stable?
2. When the robot begins to move one of its arms (arm A) to a new handhold (only 3 arms are contacting handholds), will it still be stable?
3. When arm A arrives at the new handhold (all 4 arms are again contacting handholds), is it in a stable configuration?
4. Is the positioning of arm A on this new handhold capable of keeping the robot stable once the next arm (arm B) begins to move?
5. Is the positioning of arm A on the new handhold capable of keeping the robot stable (given arm B's new position) when arm C begins to move?
6. Etc.

Conditions 1 and 3 are actually redundant. If condition 2 is true, conditions 1 and 3 must automatically be true. This chaining of stability conditions for each new arm movement ends when arm A moves to another new handhold. Figure 6 demonstrates this chaining through an example of poor low-level planning. Figure 6(a) shows the initial position of the robot. This is a stable configuration, although it is not a very good configuration due to the positioning of arm D. In Figure 6(b), arm B has moved to a new position. In Figure 6(c), arm A has also moved to a new position. Although this configuration is still stable, it is visibly precarious since arms A, B, and D are all at the same angle. As soon as arm C moves away from its handhold, the robot will no longer be stable, and will slide down and to the left. A better selection of movements is diagrammed in Figure 7.

The complete stability planning problem involves looking all the way ahead to the goal. Searching for a feasible path to the goal may involve standard searching algorithms, such as depth or breadth first searches of a tree of moves extending back from the goal position. To improve the speed of the searching algorithm, a greedy method might be developed that only looks one step ahead.

If it is not possible to place arms into positions that will result in stability during further moves, it may be necessary for the low-level planner to shift the robot body while maintaining the existing handholds in a way that will ensure future stabilty (Figure 8).

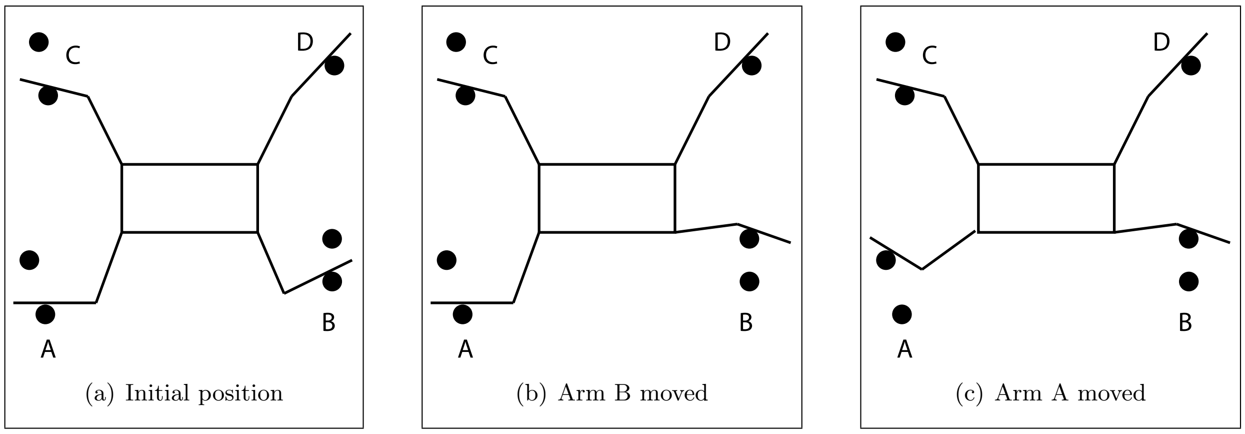


(a) Initial position

(b) Arm B moved

(c) Arm A moved; moving C results in unstable position

Figure 6: Sequence of robot positions leading to an unstable configuration



(a) Initial position

(b) Arm B moved

(c) Arm A moved

Figure 7: Sequence of robot positions with good forward-looking stability

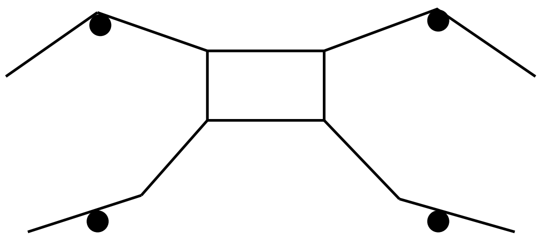


Figure 9: Position used to recenter the robot during a gait cycle

3.1 Alternate Climbing Methods

Several alternate ways exist for the robot to climb the wall. These include modifications to the robot (sensors and ratchet arms), and modifications of the wall (vertical ladder). Adding sensors to the robot would make it possible to use sensor feedback to guide the robot as it climbs the wall. The use of a vertical ladder was explored in this thesis, while the others represent possible future work.

The alternate climbing method that was examined involves a regular configuration of the wall; specifically, a ladder formed from two sets of pegs in vertical lines. In this configuration, and in any configuration that repeats as it progresses up the wall, it is possible to climb using a cyclic gait. This greatly simplifies the problem of finding a path up the wall, as it is only necessary to find one complete cycle. The length of the cycle is determined by the regularity of the wall. For a ladder with perfectly vertical sides, the cycle only needs to result in each arm of the robot being one handhold higher up the wall.

In order for a gait to be successful, the robot must not shift from side to side as it climbs up the wall. Thus, it is necessary to perform some motion or sequence of motions that returns the robot to some known configuration. For a sufficiently narrow ladder, the position shown in Figure 9 is capable of recentering the robot. In this position, the robot hooks its upper arms over the handholds at angles that cause the robot to slide until the handholds are at the elbow joints. For other configurations, it may be necessary for the robot to fall slightly in order to become recentered. These recentering methods require the robot to take some action in order to remove error from the system. Thus, the fact that the robot is climbing open-loop encourages more robust trajectories to ensure success.

There are different possibile sensing methods that can be used to provide feedback. The most comprehensive method involves a vision system, such as the one utilized by Tenzing. Vision provides complete information on the location of the robot with respect to all of the pegs. This removes many of the uncertainties involved with open-loop climbing, as the robot's internal representation of where it thinks it is can be more closely matched to where it actually is.

A much simpler sensing method involves the use of current-sensing servos. These servos provide information about how much current they are drawing. The amount of current draw increases when the servo is meeting resistance, as would be the case if the arm had run into an obstacle. With this system, it might be possible to climb a wall without any pre-existing knowledge of the wall's configuration. Essentially, the robot would be climbing blindly, swinging its arms around to find obstacles, and then using them as handholds. A major disadvantage to this system is that the robot might end up climbing up a false path that does not lead to the top of the wall. Simple touch sensors would provide less information than current sensors, but they should still yield enough information to allow the robot to feel its way up the wall.

Ratchet arms might allow the robot to be capable of climbing in any arbitrary field of handholds, especially if the robot body is raised away from the wall to a sufficient distance that it will not collide with handholds. The basic idea of a rachet arm is an arm that can push past handholds in one direction, but not in the other. One possible method of accomplishing this is to use spring-loaded arms that can bend downward to allow the arm to push up past a handhold, but that are prevented from bending upward to allow the arm to rest on top of a handhold. Another possible method involves attaching a sloped plane to the upper portion of the arm. This would cause the handhold to push the arm outward as the servos move the arm up-

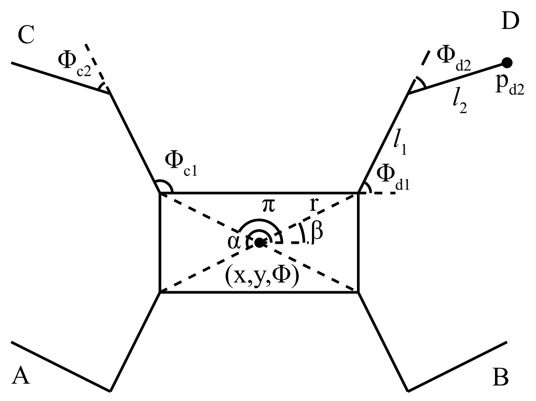


Figure 10: Variables used in the forward kinematics calculation

wards. The current robot construction should have enough flexibility in this dimension to make this possible, but a spring-loaded system might be necessary. If the arbitrary field of handholds is dense enough, a robot with ratchet arms should be able to climb blindly using a cyclic gait.

4 Mathematical Analysis

Mathematical analyses were performed to determine the forward and inverse kinematics of the robot and to examine the stability of the robot in different configurations. The forward kinematics equations are used in the GUI that controls the robot. The stability calculations are currently used as a tool for designing gaits. Future work would involve combining the inverse kinematics and stability calculations as part of the automatic planning software.

4.1 Forward Kinematics

Forward kinematics (FK) maps the configuration of the climbing robot, given below as q, to the (x, y) positions of the endpoints of the arms. It is also possible to compute the (x, y) positions of the joints as intermediate steps in the overall FK calculation if desired. Figure 10 indicates the physical meanings of the variables used in computing the forward kinematics.

The configuration of the climbing robot is given by

$$q = \begin{pmatrix} x \\ y \\ \theta \\ \theta_{a_1} \\ \theta_{a_2} \\ \theta_{b_1} \\ \theta_{b_2} \\ \theta_{c_1} \\ \theta_{c_2} \\ \theta_{d_1} \\ \theta_{d_2} \end{pmatrix}, \tag{1}$$

where x, y, and θ are the position and rotation of the robot's body, and the remaining θ_{i_n} values represent the angles of the various arm servos. i represents the arm, where A is the lower left arm, B is the lower right, C is the upper left, and D is the upper right. The number n represents whether the servo is the inner servo ($n = 1$) or the outer servo ($n = 2$).

Let $c_{ijk\cdots}$ and $s_{ijk\cdots}$ represent $\cos(\theta_i + \theta_j + \theta_k + \cdots)$ and $\sin(\theta_i + \theta_j + \theta_k + \cdots)$, respectively. Taking arm D as an example, we see that

$$p_{d_0} = \begin{pmatrix} x_{d_0} \\ y_{d_0} \end{pmatrix} = \begin{pmatrix} x + rc_{\theta\delta} \\ y + rs_{\theta\delta} \end{pmatrix} \tag{2}$$

$$p_{d_1} = \begin{pmatrix} x_{d_1} \\ y_{d_1} \end{pmatrix} = \begin{pmatrix} x + rc_{\theta\delta} + l_1 c_{\theta d_1} \\ y + rs_{\theta\delta} + l_1 s_{\theta d_1} \end{pmatrix} \tag{3}$$

$$\begin{aligned} p_{d_2} &= \begin{pmatrix} x_{d_2} \\ y_{d_2} \end{pmatrix} && (4) \\ &= \begin{pmatrix} x + rc_{\theta\delta} + l_1 c_{\theta d_1} + l_2 c_{\theta d_1 d_2} \\ y + rs_{\theta\delta} + l_1 s_{\theta d_1} + l_2 s_{\theta d_1 d_2} \end{pmatrix} && (5) \\ &= R_\theta \begin{pmatrix} x + rc_{\delta} + l_1 c_{d_1} + l_2 c_{d_1 d_2} \\ y + rs_{\delta} + l_1 s_{d_1} + l_2 s_{d_1 d_2} \end{pmatrix} && (6) \end{aligned}$$

R_θ is defined as $\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$. This is a rotation matrix, which has the effect of rotating the x, y coordinates by θ.

These equations can be combined into the following form to describe the endpoints of all 4 arms at once (each 0 represents a 2x2 matrix of zeros):

$$\mathbf{R}_\theta = \begin{pmatrix} R_\theta & 0 & 0 & 0 \\ 0 & R_\theta & 0 & 0 \\ 0 & 0 & R_\theta & 0 \\ 0 & 0 & 0 & R_\theta \end{pmatrix} \tag{7}$$

$$p_2 = \mathbf{R}_\theta \cdot \begin{pmatrix} x + rc_\alpha + l_1 c_{a_1} + l_2 c_{a_1 a_2} \\ y + rs_\alpha + l_1 s_{a_1} + l_2 s_{a_1 a_2} \\ x + rc_\beta + l_1 c_{b_1} + l_2 c_{b_1 b_2} \\ y + rs_\beta + l_1 s_{b_1} + l_2 s_{b_1 b_2} \\ x + rc_\gamma + l_1 c_{c_1} + l_2 c_{c_1 c_2} \\ y + rs_\gamma + l_1 s_{c_1} + l_2 s_{c_1 c_2} \\ x + rc_\delta + l_1 c_{d_1} + l_2 c_{d_1 d_2} \\ y + rs_\delta + l_1 s_{d_1} + l_2 s_{d_1 d_2} \end{pmatrix} \tag{8}$$

p_1 and p_0 are similarly defined for the elbow and shoulder joints, respectively.

4.2 Inverse Kinematics

Inverse kinematics (IK) maps the (x, y) position of a point on the arm to the joint angles that will position the arm at the desired (x, y) position. Since the arm doesn't need to contact a handhold at the end of the arm (it is actually undesirable to do so), the IK equations have been modified to give the range of angles for which contact with a handhold occurs.

Since all four arms are symmetrical, the following derivation only considers one of the arms, which is modeled as a 2R arm with a fixed base. The configuration of the arm is given by

$$q = \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix} \tag{9}$$

The links in the arm have lengths l_1 and l_2. For what ranges of θ_1 and θ_2 is there a collision between l_2 and a point A at (x, y)? To reduce the number of solutions, only those values for which $0 \le \theta_1 \le \pi$ and $0 \le \theta_2 \le \pi$ are considered. Heuristically, the climbing methods that yield the most stability have these constraints.

Let u be the distance from the second revolute joint (the elbow) along l_2 to the point of collision. Given the joint constraint, the minimum value of u occurs when $\theta_2 = 0$ and $\theta_1 = \arctan\left(\frac{y}{x}\right)$. This value is $u_{\min} = \sqrt{x^2 - y^2} - l_1$.

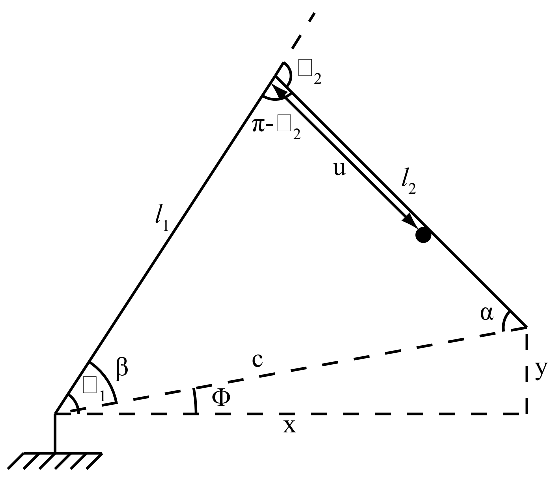


Figure 11: Inverse Kinematics Geometry

The maximum value of u is clearly l_2, at which point the end of the arm will be touching point A. The following standard inverse kinematics calculation gives θ_1 and θ_2 for this configuration:

Consider the geometry in Figure 11, with $c = \sqrt{x^2 + y^2}$. By the Law of Cosines, $c^2 = a^2 + b^2 - 2ab\cos(C)$, we have

$$x^2 + y^2 = l_1^2 + l_2^2 - 2 l_1 l_2 \cos(\pi - \theta_2) \tag{10}$$

$$\frac{x^2 + y^2 - l_1^2 - l_2^2}{-2 l_1 l_2} = \cos(\pi - \theta_2) \tag{11}$$

$$\theta_2 = \arccos\left(\frac{x^2 + y^2 - l_1^2 - l_2^2}{2 l_1 l_2}\right) \tag{12}$$

Because of the joint constraint on θ_2, one of the two solutions produced by the above calculation can be discarded. To compute θ_1, the Law of Cosines is applied again.

$$l_2^2 = x^2 + y^2 + l_1^2 - 2 l_1 \sqrt{x^2 + y^2} \cos(\beta) \tag{13}$$

$$\frac{l_2^2 - l_1^2 - x^2 - y^2}{-2 l_1 \sqrt{x^2 + y^2}} = \cos(\beta) \tag{14}$$

$$\beta = \arccos\left(\frac{l_2^2 - l_1^2 - x^2 - y^2}{-2 l_1 \sqrt{x^2 + y^2}}\right) \tag{15}$$

Once again, due to the geometry, one of the two solutions for β can be discarded. From the figure, we have $\theta_1 = \phi + \beta$. Thus:

$$\left.\begin{array}{r} \theta_1 = \beta + \operatorname{arctan2}(y, x) \\ \theta_2 = \arccos\left(\frac{x^2 + y^2 - l_1^2 - l_2^2}{2 l_1 l_2}\right) \end{array}\right\} \Rightarrow u_{\max} = l_2 \tag{16}$$

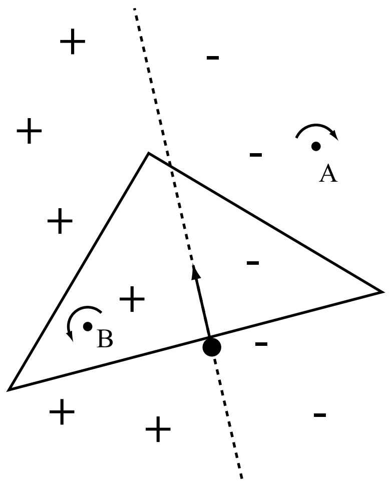


Figure 12: Reuleaux's method applied to one contact point.

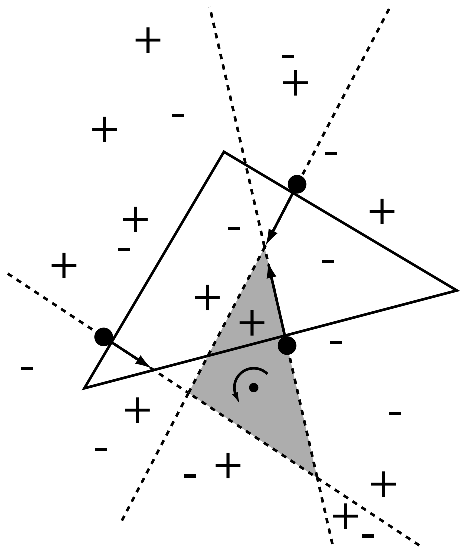


Figure 13: Reuleaux's method applied to 3 contact points.

$$\left.\begin{array}{r}\theta_1 = \arctan2\left(y, x\right) \\ \theta_2 = 0\end{array}\right\} \Rightarrow u_{\min} = c - l_1, \quad (17)$$

where $\arctan2(y, x)$ is the 2-argument arctangent function provided by standard math libraries in C and Java. This function takes into account the signs of y and x when computing the arctangent.

If l_1 is replaced with $u = [\sqrt{x^2 + y^2} - l_1, l_2]$ in equation 16, it results in a range of θ_1, θ_2 values for which contact with point A occurs somewhere along the outer arm.

4.3 Stability

We analyze the stability of the robot using two methods, one geometric, and one algebraic. Both methods describe the possible free motions that the system can exhibit. The first method, Reuleaux's Method, is a graphical method. The second method is an algebraic method involving the normals at the contact points.

4.3.1 Reuleaux's Method

Reuleaux's method [7] is a graphical method that uses the contact normals to determine free motions. This method is based on the well-known theorem, mentioned in Mason [6], which states that every instantaneous motion of a planar rigid body can be described as a differential rotation about an instantaneous rotation center, possibly at infinity.

As an example, consider the basic case given in Figure 12. The triangle has one point contact, with the normal as shown. Extending the normal divides the space into two half-planes. For any rotation center in the right half-plane (such as point A), only negative rotations are possible. For any rotation center in the left half-plane (point B), only positive rotations are possible. As each new contact point is added, it is considered in the absence of the other contact points. Once all of the positive and negative rotation regions have been identified, only the regions with only positive or only negative rotations will allow free motions that do not cause penetration. If two more contact points are added to the triangle example (Figure 13), the free motions are reduced to only positive rotations in the shaded region.

Adding gravity to the system is a simple step. Gravity can be thought of as a point contact that prevents upward motion of the center of mass. Thus, it can be drawn by a downward normal at the center of mass. Applying all of these concepts, it is possible to determine the possible rotation centers for the configuration of the climb-

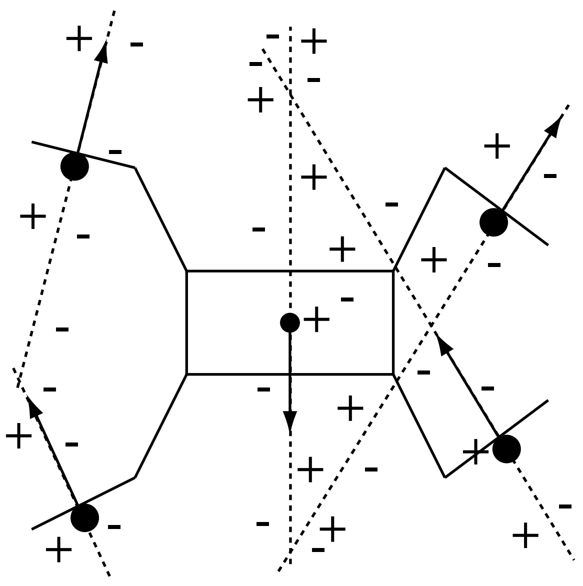


Figure 14: Reuleaux's method applied to the robot.

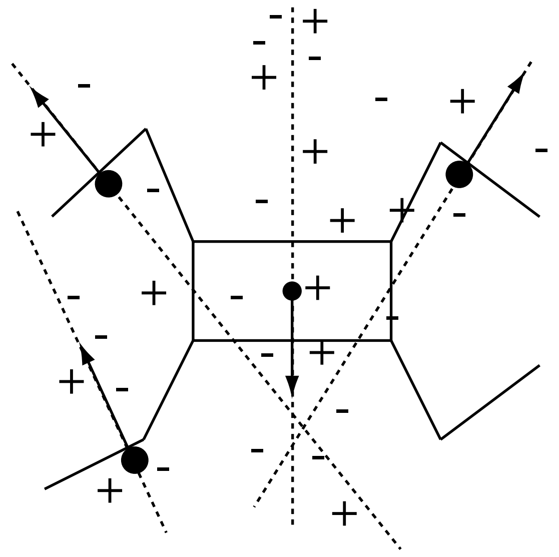


Figure 15: The robot is stable with only 3 handholds.

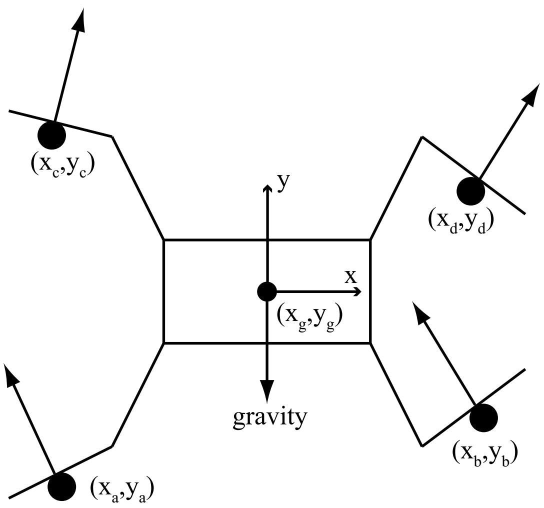


Figure 16: Stability Geometry

ing robot given in Figure 14. As shown in the figure, there are no possible rotation centers. It can also be shown that the robot can be stable with only 3 handholds and gravity (Figure 15).

4.3.2 Algebraic Method

Although Reuleaux's method is very descriptive about possible free motions, it is difficult to implement in code. Therefore, it was necessary to compute possible free motions using an algebraic method as well. The basic concept of the algebraic method for calculating the possible free motions of a system involves comparing the velocity vectors of the contact points to the normal vectors of the contacts. If the velocity is in the half-plane formed by the normal, then the velocity is permissible. For this derivation, Figure 16 details the variables and geometry.

This calculation assumes that the robot is a rigid body, and that it is capable of locking its arms such that they do not move at all. The configuration of the robot is given by

$$q = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \tag{18}$$

Let $\mathbf{x}$ be a vector that collects all of the contact points:

$$\mathbf{x} = \begin{pmatrix} x_a \\ y_a \\ x_b \\ y_b \\ x_c \\ y_c \\ x_d \\ y_d \\ x_g \\ y_g \end{pmatrix} \tag{19}$$

$\mathbf{x_g} = (x_g, y_g)$ represents the "contact point" of the gravitational force. The coordinate system is defined to have its origin at the center of mass (the weight of the arms is neglected), and thus $\mathbf{x_g} = (0, 0)$.

Additionally, define the normal vectors as follows, and collect them into a matrix N (each 0 in N is a 2-vector of zeros):

$$\mathbf{n_a} = \begin{pmatrix} \sin(\theta + \theta_{a_1} + \theta_{a_2}) \\ -\cos(\theta + \theta_{a_1} + \theta_{a_2}) \end{pmatrix} \tag{20}$$

$$\mathbf{n_b} = \begin{pmatrix} -\sin(\theta + \theta_{b_1} + \theta_{b_2}) \\ \cos(\theta + \theta_{b_1} + \theta_{b_2}) \end{pmatrix} \tag{21}$$

$$\mathbf{n_c} = \begin{pmatrix} \sin(\theta + \theta_{c_1} + \theta_{c_2}) \\ -\cos(\theta + \theta_{c_1} + \theta_{c_2}) \end{pmatrix} \tag{22}$$

$$\mathbf{n_d} = \begin{pmatrix} -\sin(\theta + \theta_{d_1} + \theta_{d_2}) \\ \cos(\theta + \theta_{d_1} + \theta_{d_2}) \end{pmatrix} \tag{23}$$

$$\mathbf{n_g} = \begin{pmatrix} 0 \\ -1 \end{pmatrix} \tag{24}$$

$$N = \begin{pmatrix} \mathbf{n_a}^T & 0 & 0 & 0 & 0 \\ 0 & \mathbf{n_b}^T & 0 & 0 & 0 \\ 0 & 0 & \mathbf{n_c}^T & 0 & 0 \\ 0 & 0 & 0 & \mathbf{n_d}^T & 0 \\ 0 & 0 & 0 & 0 & \mathbf{n_g}^T \end{pmatrix} \tag{25}$$

We will compute the free motions of the robot, $\dot{q}(t)$, as a function of $q(t)$. $\mathbf{x} = f(q)$, where f is the forward kinematics map, as calculated in Section 4.1. Then, taking the time derivative, $\dot{x} = J(q)\dot{q}$, where J is the Jacobian matrix of partials of f with respect to q. Rather than compute these partials directly, we use the "cross-product method" [2] for calculating the Jacobian:

$$J = \begin{pmatrix} \hat{\mathbf{x}} & \hat{\mathbf{y}} & \hat{\mathbf{z}} \times \mathbf{x_a} \\ \hat{\mathbf{x}} & \hat{\mathbf{y}} & \hat{\mathbf{z}} \times \mathbf{x_b} \\ \hat{\mathbf{x}} & \hat{\mathbf{y}} & \hat{\mathbf{z}} \times \mathbf{x_c} \\ \hat{\mathbf{x}} & \hat{\mathbf{y}} & \hat{\mathbf{z}} \times \mathbf{x_d} \\ \hat{\mathbf{x}} & \hat{\mathbf{y}} & \hat{\mathbf{z}} \times \mathbf{x_g} \end{pmatrix} \tag{26}$$

$\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, and $\hat{\mathbf{z}}$ are the direction vectors. Because $\mathbf{x_g} = (0, 0)$, the term $\hat{\mathbf{z}} \times \mathbf{x_g}$ is zero. Computing $N \cdot J$ yields a matrix that converts velocities of the configuration variables to velocities of the contact points in the direction of the normals.

$$\begin{aligned} J_N &= N \cdot J && (27) \\ &= \begin{pmatrix} n_{a_x} & n_{a_y} & \mathbf{n_a}^T \cdot (\hat{\mathbf{z}} \times \mathbf{x_a}) \\ n_{b_x} & n_{b_y} & \mathbf{n_b}^T \cdot (\hat{\mathbf{z}} \times \mathbf{x_b}) \\ n_{c_x} & n_{c_y} & \mathbf{n_c}^T \cdot (\hat{\mathbf{z}} \times \mathbf{x_c}) \\ n_{d_x} & n_{d_y} & \mathbf{n_d}^T \cdot (\hat{\mathbf{z}} \times \mathbf{x_d}) \\ n_{g_x} & n_{g_y} & 0 \end{pmatrix} && (28) \end{aligned}$$

Thus, $J_N\dot{q}$ represents the velocities of the contact points along the normals. If $J_N\dot{q} \geq 0$, then motion is possible along the contact normals and along the "virtual contact" due to gravity. if $J_N\dot{q} \leq 0$, for any $\dot{q}$, free motion is not possible, and the configuration is stable.

The above form ($Ax \leq 0$) is known as the face-normal representation of a polyhedral convex cone (PCC). In this form, each row a_i of A is a normal to a half-plane which forms a half-space in 3-dimensional space. The intersection of all of these half-spaces is the PCC. If x (in this case, $\dot{q}$) lies in the PCC, then it is a valid solution to the inequality. For the purposes of computing stability, it is necessary for the PCC to contain only $\dot{q} = 0$.

It can be determined if a configuration is stable by testing to see if the PCC is empty. It is possible to convert the face-normal representation to an edge-normal representation [3, 4], from which it can be determined if the edges contain only the origin. Finding stable configurations is more difficult, as it is necessary to determine how changing the configuration will affect the PCC. For the

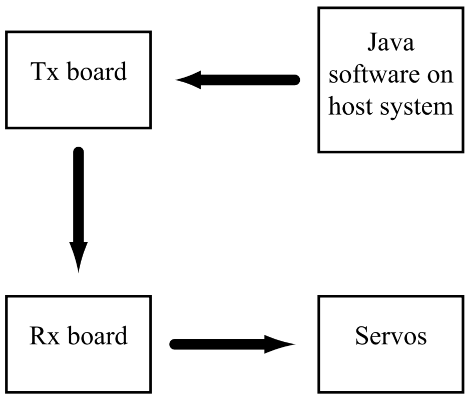


Figure 17: Block diagram of overall system

climbing robot, the PCC has 5 faces. Four faces are sufficient to reduce the PCC to just the origin. This should make it slightly easier to find a stable configuration. If we had infinite computation capabilities, the best way to find a configuration yielding an empty PCC would be to start with one arm at some given position, and to choose the positions of the remaining arms such that each new arm reduced the PCC by as much as possible. Essentially, we are sampling to find a PCC that is empty. With less computational power, a probabilistic method will be necessary.

5 Robot Control

As described earlier, the robot is controlled through IR communications. This relies on three main pieces of software (Figure 17). The host system runs a Java application that sends commands through a serial cable to the transmitter SV203C (Tx board). Code written in SVBASIC runs on the Tx board, and retransmits the commands along with a checksum to the receiving SV203C (Rx board). Additional SVBASIC code on the Rx board interprets and verifies the commands before moving the appropriate servo.

5.1 Embedded System Code

The SV203C boards contain microcontrollers that can be programmed in SVBASIC. The SVBASIC code is converted to assembly by an assembler on the host system, and the corresponding hex file is downloaded to the SV203 through a serial cable. Both boards are set to automatically start running their respective program when they are powered up.

5.1.1 Tx Board

Code 1 Pseudocode for the Tx board

```
• Wait for memory location 186 to become 1 //This is changed by serial commands
• Transmit memory location 184 (ServoNumber) via IR //Set by serial
• Transmit memory location 185 (ServoPosition) via IR //Set by serial
• CheckSum = ServoNumber + ServoPosition
• Transmit CheckSum via IR
• Reset memory location 186 to 0
```

The pseudocode for the Tx board is listed in Code 1. It is not possible to command the SV203C to send IR data directly through serial commands. Thus, it was necessary to write code to monitor a specific memory location (which can be set by a serial command). The host system should write the servo number and position first, and then set memory location 186 to 1. This triggers the transmission of 3 bytes of data to the robot (Servo number, servo position, and checksum).

5.1.2 Rx Board

The pseudocode for the Rx board is listed in Code 2. This code needs to perform some basic error checking to make sure that the IR data was correctly received. Early tests of the IR system demonstrated that bad bytes do occasionally get received, which caused the servos to jump randomly. The checksum was implemented to prevent this from happening.

The Rx board functions as a small state machine, and deals with the received IR byte according to the state. It expects to receive a servo number, servo position, and a checksum. If the transmission gets out of sync, either the servo number validity check or the checksum verification will eventually resync the Tx and Rx boards.

Code 2 Pseudocode for the Rx board

```
• Set State = 0
• IRWait: Wait until IR data is received
• If State = 0
    – If the received byte is a valid servo number ($1 \leq$ServoNumber$\leq 8$), then save it and set State = 1
    – Go to IRWait
• If State = 1 Then save the received byte as the servo position, set State = 2, and go to IRWait
• If State = 2
    – ChecksumExpected = ServoNum + ServoPos
    – If ChecksumExpected matches the received byte, move ServoNum to ServoPos
    – In either case, set State = 0
    – Go to IRWait
```

5.2 Host System Code

The host system runs a Java program that provides a GUI interface to the robot. The first revision of the program only provided sliders that controlled the servos. This was replaced with a version that used the forward kinematics equtions to draw a pictorial representation of the robot's current position. The robot's arms are controlled by selecting a servo and using the up and down arrows on the keyboard to provide smooth motions.

The host software provides the capability to record keyframes to a file, and to later play them back without the delays between motions that result from planning the next move during manual operation. A significant advantage of the keyframe system is that it makes it possible to shift all of the servos together (as shown earlier in Figure 8), which is difficult to do during manual control. This is done by looping over all 8 servos. If any servo is not at the position of the next keyframe, it is incremented or decremented by 1, as appropriate. This is not true interpolated movement, as some servos may arrive at their goal positions much sooner than others. Moving multiple servos at once requires a longer delay between serial commands than moving a single servo. The reason for this has not been determined; it is possible that the SV203C takes extra time to switch between controlling different servos.

6 Results and Conclusions

The robot is capable of climbing under manual control and on the basis of a set of pre-recorded keyframes. In addition, a cyclic gait has been developed for climbing a vertical ladder configuration of the wall.

To reiterate, the three main climbing modes are

1. Manual remote control
2. Autonomous, with pre-recorded keyframes
3. Autonomous, using a simple cyclic gait

In manual control mode, the robot is directly controlled by the human operator. The operator manipulates the robot through the sliders in the Java GUI. Commands are sent to the Tx board through the serial port, and then on to the Rx board through IR. This is the slowest method of control, primarily because it takes time for the operator to decide which arm should be moved next. However, this mode is very reliable, as there is an inherent feedback system through the operator's vision.

I have successfully used this manual control system to climb the wall pictured in Figure 4. Through doing this, I was able to get a good feel for stable positions, and to use this for later implementation of the gait. A major benefit of manual control is that some sideways motion was possible. I could use an arm to push or pull sideways on a handhold to slide the robot, which would be difficult to do automatically due to the effects of friction.

During one of the climbs on the wall, I recorded the keyframes using the recording function of the GUI. It was easily possible to manually modify and splice together the files containing the recorded keyframes. This made it possible to create keyframe recordings piece by piece. I could climb some distance, and if the robot got into a difficult position, I could remove the last few keys, play back the good keys, and start recording from the ending position of the previous set of keys. During these pre-recorded climbs, it was necessary to be very precise in positioning the robot on the wall in its initial position. The robot needed to be within about 0.5 cm of the initial position to successfully climb the wall. Through this process, I was able to make the robot climb up 2 rows of handholds using a single set of about 60 keyframes. This also assisted in later development of the gait, as I was able to observe one of the recentering motions that was later used in the gait.

With the wall in a perfectly vertical ladder configuration, the robot can climb using a cyclic gait (Figure 18). The first cycle of the gait was manually operated and recorded. The resulting keyframes were then cycled eight times, enabling the robot to climb to the top of the wall in about 10 minutes. This yields an upward speed of about 0.8 pegs per minute.

The gait begins at the bottom of Figure 18 and proceeds upward. The robot begins by lifting the left leg and then the right leg onto higher handholds. In the process of moving the right leg, it is necessary to move the right arm to lift the entire robot upward slightly. Next, the robot performs the recentering move (center image). In this move, it hooks the arms over the pegs, and lifts the entire body. Then, the left and right arms are moved to higher handholds. Finally, the robot returns all of the joints to their initial positions so that the cycle can repeat itself.

A toy rock climbing robot was successfully developed. Although it is not capable of truly autononmous climbing, it can climb a ladder using a cyclic gait in an open-loop mode. It achieves this through the use of a recentering motion that ensures that the robot is correctly positioned during each cycle. Currently, this recentering occurs during the middle of the cycle, and as such, some care is required in the initial positioning of the robot. If the robot were produced as a kit marketed to the general public, it would probably be necessary to perform the recentering at the start of the climb. The robot is sufficiently simple that it could be marketed as a kit. The robot does not require a bulky tether to operate correctly, and the GUI is fairly easy to use.

This thesis has also explored the mathematics involved in robotic climbing, including the forward and inverse kinematics, and calculations of stability. In the future, these could be integrated into a full path planning algorithm to make the robot completely autonomous. Various high and low-level path planning strategies have been examined. The cyclic gait method has been suc-

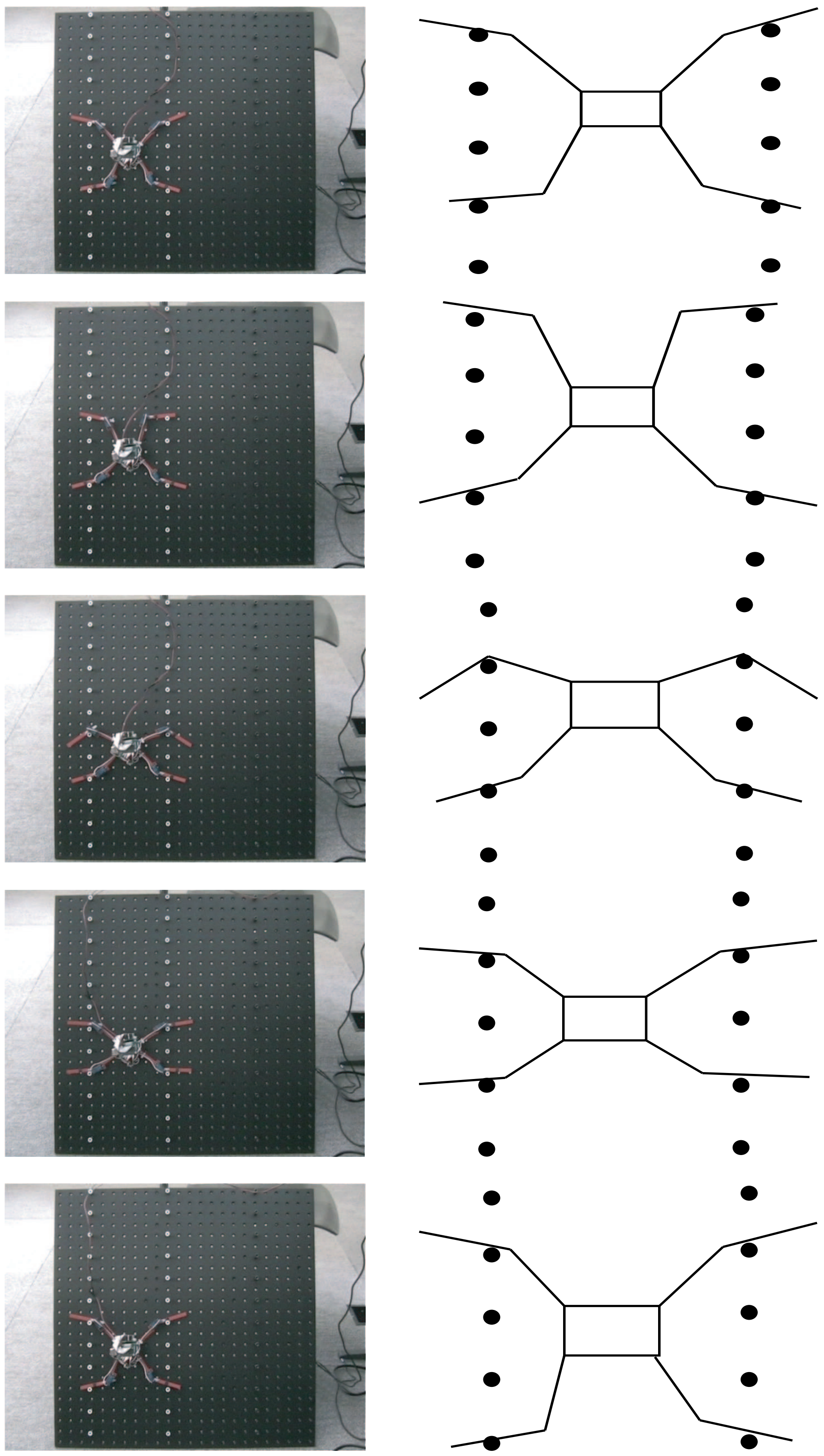


Figure 18: Sequence of robot configurations during a single cycle of a gait

cessfully implemented for one wall configuration, and other gaits could be developed for other regular configurations. It is possible that a general gait could be developed for a set of similar configurations, provided that a generic recentering strategy exists.

There are several potential areas of further exploration. A full path planning algorithm could be developed to allow the robot to climb any wall based only on knowledge of the locations of the pegs. A general-purpose cyclic gait could exist for a certain set of wall configurations. Sensors could be added and incorporated into the climbing algorithm, and the robot itself could be redesigned to allow it to climb blindly without any knowledge of the wall. The work contained in this thesis forms the basis for these areas of further research.

7 Acknowledgments

I would like to express my sincere gratitude to my advisor, Devin Balkcom, for developing the initial thesis idea and for building the basic form of the robot. His help with understanding the mathematical basis for the climbing robot was also very valuable. I would also like to thank Stephen Linder for his suggestions and his assistance with getting the Pontech boards to work. Finally, I would like to thank Anne Loomis, Nelson Rosa, Joseph Pechter, and my parents for reviewing early versions of this thesis.

References

[1] T. Bretl, S. Rock, J. C. Latombe, Brett Kennedy, and Hrand Aghazarian. Free-climbing with a multi-use robot. In *International Symposium on Robotics Research*, 2004.

[2] John J. Craig. *Introduction to Robotics: Mechanics and Control.* Addison-Wesley, second edition, 1989.

[3] A. J. Goldman and A. W. Tucker. Polyhedral convex cones. In H. W. Kuhn and A. W. Tucker, editors, *Linear Inequalities and Related Systems*, pages 19–40. Princeton Univ., York, 1956.

[4] S. Hirai. *Analysis and Planning of Manipulation Using the Theory of Polyhedral Convex Cones.* PhD thesis, Kyoto University, March 1991.

[5] Stephen Paul Linder, Edward Wei, and Alexander Clay. Robotic rock climbing using computer vision and force feedback. In *IEEE International Conference on Robotics and Automation*, 2005.

[6] Matthew T. Mason. *Mechanics of robotic manipulation.* MIT Press, 2001.

[7] Franz Reuleaux. *The Kinematics of Machinery.* MacMillan, 1876. Reprinted by Dover, 1963.

[8] Jizhong Xiao, Jun Xiao, and Ning Xi. Minimal power control of a miniature climbing robot. In *IEEE/ASME International Conference on Advanced Intelligence Mechatronics*, pages 616–621, July 2003.