Design and Implementation of Programs for an Autonomous Robotic Arm

NSF Summer Undergraduate Fellowship in Sensor Technologies
Xavier Hanson-Lerma, SUNFEST Fellow (Electrical & Systems Engineering) – University of Pennsylvania

Advisor: Dr. Daniel Lee (Electrical & Systems Engineering)

ABSTRACT

Terrestrial robots used in reconnaissance or disaster-relief encounter obstacles throughout their missions, which necessitates the use of robotic arms to manipulate the objects that obstruct them. Over the course of ten weeks, the author participated in a research project which had the development of a control program for such an arm as its goal. This paper describes a set of programs that were designed to aid the user in removing obstacles from the path of an Unmanned Ground Vehicle (UGV) autonomously.
The programs use LIDAR data to plot a robot's surroundings, and then identify the nearest object that the arm can move. The coordinates of that object are then sent to an inverse kinematics program, which determines the best path for the robotic arm, with no additional assistance from the user.
This paper details the techniques used to characterize and implement these programs, along with the mathematics used in the inverse kinematics needed for controlling said arms.

Table of Contents

1. INTRODUCTION

The use of robots in disaster relief and reconnaissance is becoming widespread. As more of these devices are used in the field, practical concerns begin to emerge. The environments that the robots are sent in to observe will inevitably provide obstacles that need to be negotiated or removed. One way to solve this problem is to use agile flying robots, but the operating range of such devices is limited by the weight of their batteries. Creating ground devices that are capable of clearing obstructions is another (perhaps more elegant) solution, given that such devices are capable of carrying more power sources.

Comment [MW1]: Better without this comma.

Another practical concern is how much the user has to control the robot. The process of collecting information remotely should be made as simple and as efficient as possible. This can be achieved by having all or most of the functions of the robot performed autonomously, including clearing obstacles. This is done by using Light Detection and Ranging (LIDAR) to create a map of the robot's surroundings, and creating programs that use this map to assist in making decisions in obstacle clearing. The robot will make its own decisions as to how the arm will orient itself given the circumstances. This involves creating a number of inverse kinematics algorithms for the arm, which has four degress of freedom (DOF).

Plotting the robots' surroundings in three dimensions is on the cutting edge of robotics and research, and full three-dimensional imaging cannot be achieved in the small ammount of time allotted for this project (ten weeks). Therefore, we decided to have the LIDAR scanner look for obstructions in a plane parallel to the ground (Refered to as the XY Plane in this paper). Our test obstructions or "targets" were tall cylinders, whose position in the XY plane would be the same at both the LIDAR sensor's height and in the arm's range.

Our goal is simply to have the arm go to the nearest obstruction and move it aside. The computer attached to the robot arm runs the inverse kinematics programs needed to make decisions as to how the arm will position itself towards the target.

Comment [MW2]: Again, clarify the time frame.

2. BACKGROUND

2.1 ROBOTIC ARMS

Mechanical arms have been central to industry since 1961 [1]. From their inception, they were used for repetitive tasks, often in hazardous environments. Since those early days, advancements have been made in computing and miniaturization that allow the creation of devices capable of responding to their own environments, rather than repeat the same task regardless of the environmental conditions.

Comment [MW3]: Second sentence: 'From their inception' cries out for 'have been used.' Compare the next sentence for verb time. However, that next sentence implies that you were around in 1961. Puh-leeze!

2.2 LIDAR

There are many ways for the robot to perceive depth and distance, including using cameras in a stereo-vision configuration [2]. Unfortunately, this system only works with visible light, and in extreme environments this may not always be available. We thought it best to use LIDAR, a system that works by sending out a laser-light pulse and measures the amount of time it takes to bounce back [3]. This works well in low-light settings.

Comment [MW4]: So if you're happy saying 'we' here, why not in the Abstract? Why 'the author'?

2.3 OUR PLATFORM: MAGIC 2010

Fig.1 MAGIC 2010 UGV [5]

We are fortunate to already have a robot in our possession capable of mapping its surroundings using LIDAR (Fig.1). It is a compact Unmanned Ground Vehicle (UGV) developed as part of the American and Australian defense departments' MAGIC 2010 challenge [4]. It uses two Hokuyo LIDAR detectors. One of them is kept fixed on the robot, and is used for horizontal 2-D mapping. Another is mounted vertically on the front of the robot and pans side to side. This latter LIDAR detector is used for detecting the characteristics of the terrain in the robot's path. Simultaneous localization and mapping (SLAM) algorithms generate a map for the robot using data from the two LIDAR detectors. The robot uses this data to help plan the best route.

The UGVs were designed to work in a team with several others like them, but are capable of mapping their surroundings on their own [5]. For our tests, we mounted the arm on one such UGV, and used its horizontal scanning LIDAR to detect objects.

Comment [MW5]: still the proposal…

3. METHODS

3.1 THE ROBOTIC ARM AND ITS DESIGN

Rather than design a robotic arm from scratch, an existing design from CrustCrawler Inc. was used (Fig.3, left). It uses seven servos, specifically Dynamixel AX-18A servos made by Robotis. AX-18A servo motors each have a microcontroller that accepts serial data from a central computer or controller. This allows parameters like speed, torque and position to be fed directly to the servos and stored and modified in the memory of the servos' microcontrollers. The "position" parameter of the servos controls the joint angle, which is what is being controlled in our inverse kinematics equations. Each servo has an ID, which is used for sending that servo specific commands from the computer. Individual commands can be sent to several servos simultaneously. In order to ensure that the instructions were sent properly, a status packet is sent back to the computer/main controller. This status packet can include data being read from the servos, such as its current position or torque (Fig.2). [6]

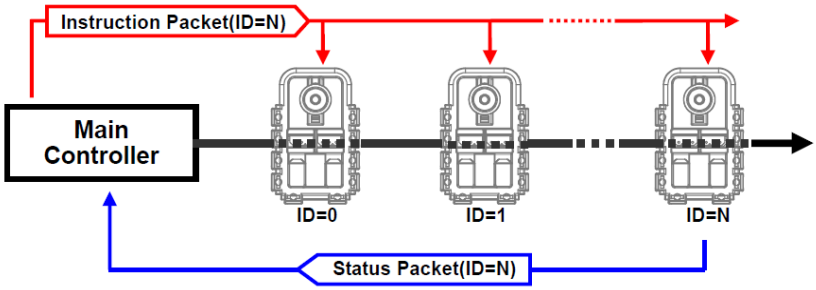

Fig. 2 Visualization of Information Sent to Group of Dynamixel Servos [6]

In our arm, the first servo operates a turntable at the base of the arm that rotates the arm. The second and third servos are located a few centimeters above the base at an angle, and their rotors face away from each other. Together they simulate an elbow. A few centimeters above that elbow is a similar joint, also controlled by a pair of servos. That joint orients the sixth servo, which is the wrist of the gripper. The wrist orients the gripper so that the flat paddles that grip the target are parallel to the target's surface. For our experiments, the targets were upright cylinders so the wrist was kept in a fixed position. The final servo opens and closes the gripper that is to grasp the targets.
In order to determine whether the target has been gripped properly, a program that reads the torque felt by the gripper servo commands the gripper to cease tightening its grip once the torque reaches a certain value, which is to be assigned experimentally.

Comment [MW6]: just 'commands'

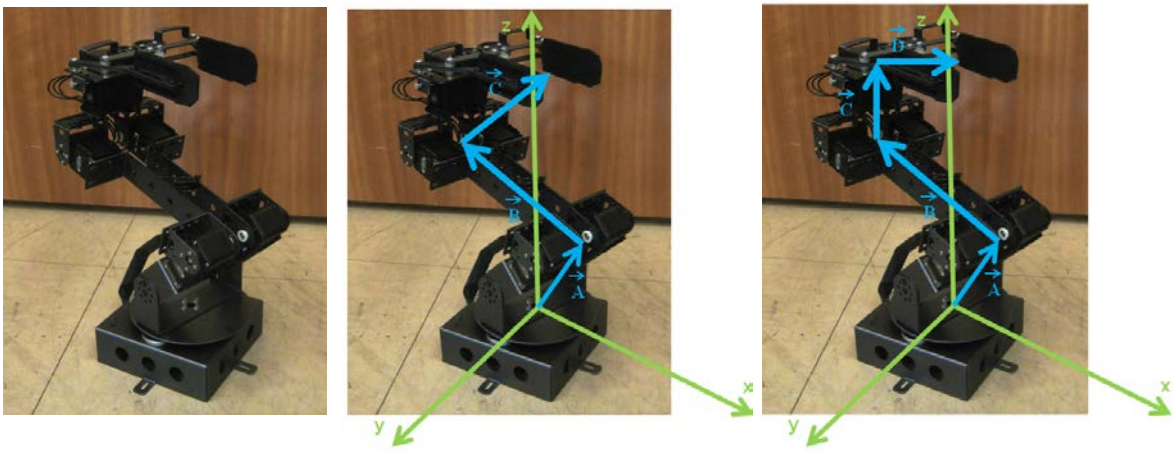

Fig.3 CrustCrawler Robotic Arm and its Three and Four Vector Representations in Three Dimensional Space Photograph and diagrams made by author

3.2 ARM KINEMATICS

Despite the number of joints, the geometry of this arm can be simplified to as few as three vectors (Fig. 3, center). **A** goes from the center of the turntable to the center of the first elbow, **B** goes between the center of that elbow to the center of the elbow above it, **C** goes between that elbow and the center of the gripper. All of these vectors have fixed lengths except for **C**, which varies depending on how open or closed the jaws of the gripper are, but its length will be known by the time inverse kinematics equations are to be evoked. More details on the gripper's kinematics are given later in this paper.
These vectors' orientations in space are defined by the angles α, θ, φ, and ψ (Fig.4). α is fixed, due to the construction of the arm. θ is defined by the "position" of Servo 1, φ by the bottom elbow (Servos 2 and 3), and ψ by the top elbow (Servos 4 and 5).
The goal of the inverse kinematics program is to orient the center of the arm's gripper to a target point, P. The vector sums of **A**, **B**, and **C** ought to add up to P's position vector **P**.

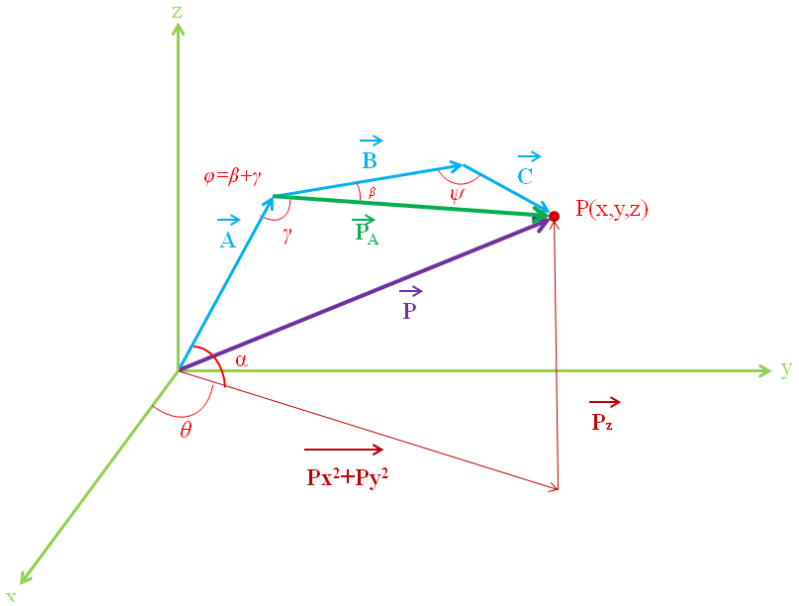

Fig.4 Three Vector Representation of Arm with $\mathbf{P_A}$ Inserted as an Aid to Calculation

The trigonometry involved in calculated the angles θ, φ, and ψ works as follows:

Given: The lengths of **A**, **B**, **C** and **P**, angle α, and P's coordinates

θ can be determined with only the x and y coordinates of point P, using the Pythagorean theorem and trigonometric identities (Fig.5, a). ψ necessitates the creation of a vector $\mathbf{P_A}$ (Fig.5, b), so that we may use the law of cosines to find it (Fig.5, c). $\mathbf{P_A}$ splits φ into β and γ (Fig.5, d) . β can be found with the law of sines (Fig.5, e)**.** γ can be determined using the definition of a vector dot product (Fig.5, f) **.**

$$\theta = \cos^{-1}\left(\frac{x}{\sqrt{(x^2 + y^2)}}\right) \qquad \mathbf{P_A} = \mathbf{P} - \mathbf{A} \qquad \psi = \cos^{-1}\left(\frac{-|\mathbf{P_A}|^2 + |\mathbf{B}|^2 + |\mathbf{C}|^2}{2|\mathbf{B}||\mathbf{C}|}\right)$$

(a) *(b)* *(c)*

$$\varphi = \beta + \gamma \qquad \beta = \sin^{-1}\left(\frac{|\mathbf{C}|\sin\psi}{|\mathbf{P}|}\right) \qquad \gamma = \cos^{-1}\left(\frac{\mathbf{P_A} \bullet -\mathbf{A}}{|\mathbf{P_A}|\,|\mathbf{A}|}\right)$$

(d) *(e)* *(f)*

Fig. 5 Formulas Used In Defining the Three Vector Inverse Kinematics Model for the Arm

Representing the arm with three vectors is adequate when one is working with a full three dimensional representation of the robot's surroundings, but our system is only plotting in two dimensions, and we need the grasper to always be parallel to the ground in order to grasp our tall target (it wouldn't do for the grasper to come to the target at an angle, because it would knock over the target. This was learnt by experiment). Therefore, we needed to add an additional constraint, which led to our four vector model (Fig.3, right and Fig.7). The vector **C** in the three vector model was replaced with $\mathbf{C_{NEW}}$ and **D**. **D** is always perpendicular to $\mathbf{C_{NEW}}$**,** and they form a right triangle with the old **C** as their hypotenuse. **D** varies in length, and depends on the grasper's position; how open or closed the "jaws" are. The movement of the centers of the grasper's fingers can be modeled with a circular path (Fig.6). **D**'s changing length is directly related to the width of the target object, which is determined from LIDAR data.

Comment [MW7]: (the)

Comment [MW8]: I'd say 'it'.

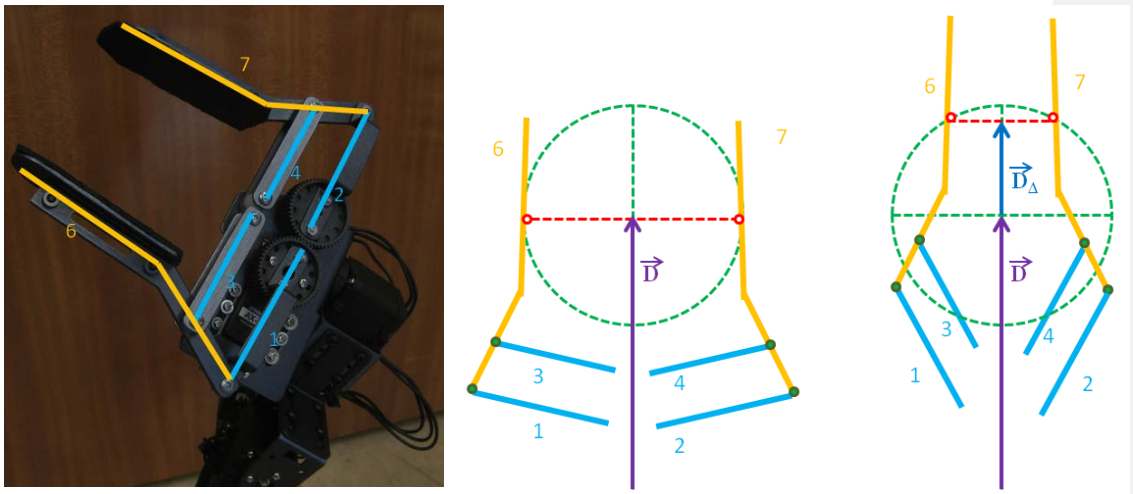

Fig. 6 Grasper Geometry.
The red circles represent the center of the grasper's fingers. D's total length is the sum of D and D_Δ

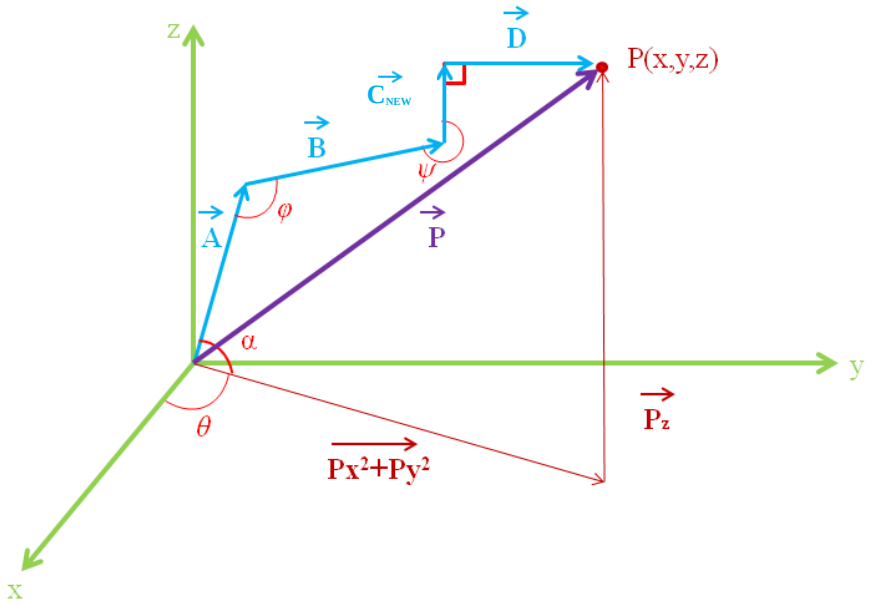

Fig.7 Four Vector Representation of Arm

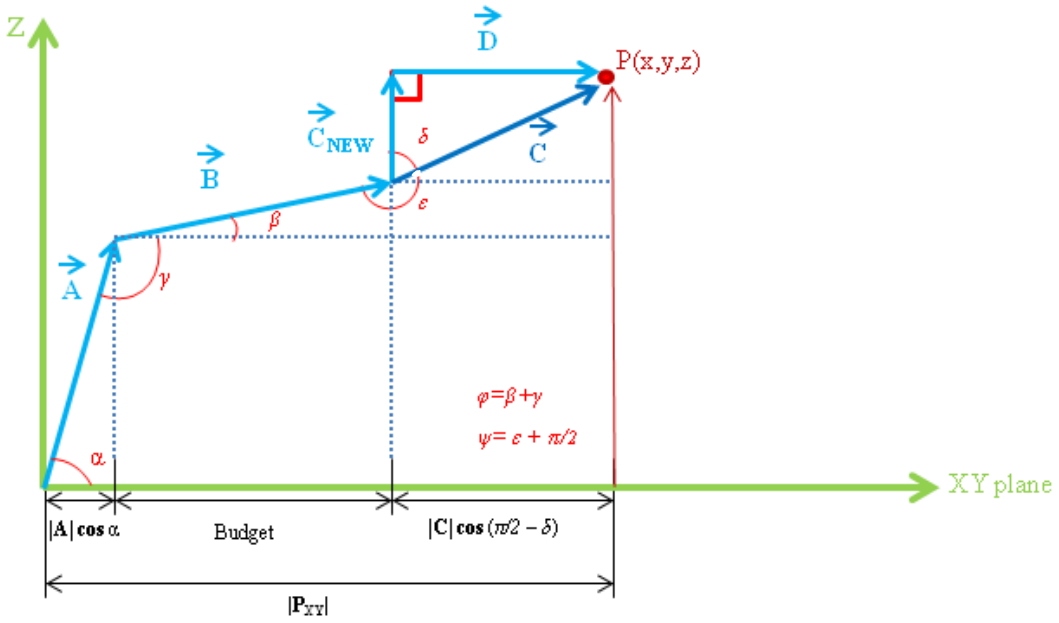

Fig. 8 Inverse Kinematics for Four Vector Model of Arm

Since only the XY plane is needed, the angles for the four vector model can be calculated as follows:

Comment [MW9]: use a hyphen to help the reader, here and elsewhere.

Given: The lengths of **A**, **B**, **C**,$\mathbf{C_{NEW}}$, **D**, angle α, and P's coordinates

θ (in Fig.7), is defined the same way in the four vector model as in the three vector model, and orients the entire arm.
We can define **A**'s shadow on the XY plane as $|\mathbf{A}| \cos \alpha$. Angle δ and **C**'s shadow on that plane can be found through trigonometry, since we are given **C**,$\mathbf{C_{NEW}}$, and **D**. Armed with these quantities, we can create a "*Budget*" that can be used to find the other angles (Fig.8, a). For ease of calculation, φ is split into angles γ and β (Fig.8, b). β can be defined with the definition of a cosine (Fig.8,c). γ and ε can be found with the properties of triangles (Fig.8, d and e respectively). Once we have ε we can find ψ (Fig.8, f).

$$Budget = |\mathbf{P_{XY}}| - |\mathbf{A}| \cos \alpha - |\mathbf{C}| \cos (\pi/2 - \delta) \quad (a)$$

$$\varphi = \gamma + \beta \quad (b)$$

$$\beta = \cos^{-1}(Budget / |\mathbf{B}|) \quad (c)$$

$$\gamma = \pi - \alpha \quad (d)$$

$$\varepsilon = \pi - \beta \quad (e)$$

$$\psi = \varepsilon + \pi/2 \quad (f)$$

Fig. 8 Formulas Used In Defining the Four Vector Inverse Kinematics Model for the Arm

3.3 LIDAR

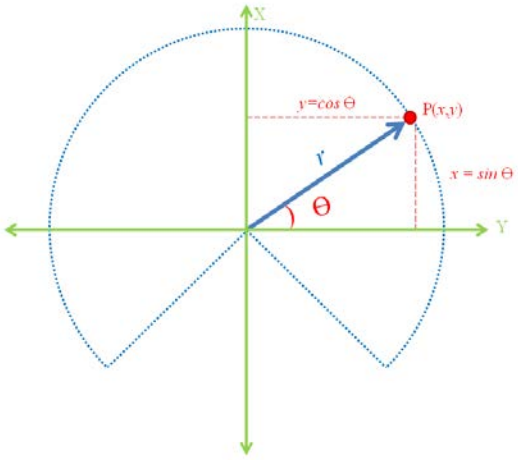

Fig.9 Representation of 2-D Plotting Using Hokuyo LIDAR Scanner. r and Θ depend on the constantly changing array of values being returned by the scanner.

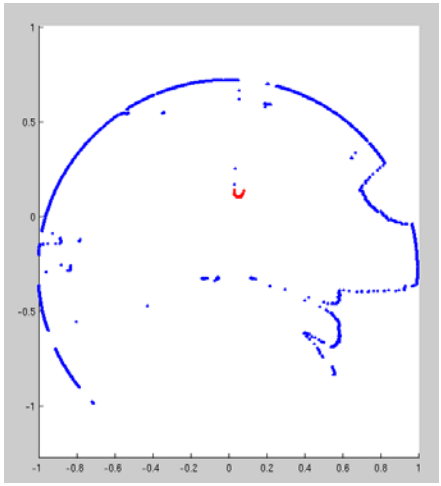

Fig.10 2-D Plot Made with LIDAR Scanner. Target is identified and highlighted in red

Our means of detecting objects is the Hokuyo UTM-30LX LIDAR sensor. It was chosen for its wide scanning range and its millimeter accuracy. The scanner returns a "distance array" containing the distances of the LIDAR sensor's surroundings at each of the angles in its range (a measurement every 0.25^{o}), 40 times a second [7]. This constantly refreshing array can be used to plot the surroundings of the robot.

Comment [MW10]: Okay; but I'd say, 'we detect objects using the….sensor.'

For our work, only the 180 ^{o} arc in front of the robot is considered, and distances that are not in not in the arm's reach are ignored. The distance array is then filtered for noise using a moving average filter.

Comment [MW11]: clunky. 180 deg. needs a plural, at least. 'The 180 degrees in front of the robot are considered;' 'Only an arc of 180 deg. is considered.'

Each index in the distance array represents an angle (Θ), and its contents represent the distance (*r*) between the scanner and the nearest object at that angle. Returning the minimum point in this matrix gives us the angle (and therefore the position) of the nearest object. Converting this polar coordinate (*r*, Θ) into the necessary Cartesian coordinates is simple (Fig. 9).

It is not enough to merely give the arm the closest point. We have to find the size of the object to be grasped to see if it is even possible for the arm's grasper to grip it, and determine the center of the object for the kinematics program.

Determining the width of the object is a matter of looking at the nearest point in the distance array, and its immediate neighbors; the differences between distances amongst these neighboring points are examined. Once the difference between one element in the distance array and its immediate neighbors is beyond a certain limit, we know that that point is an extreme or edge of the object being detected. The center of the object is the

midpoint of the line between the two extremes. Vector projection is then used to determine the width of the object.

Fig.10 shows a plot made with the filtered LIDAR data, with a target in range. The program recognized the object and highlighted it in red.

3.4 WORKING ENVIRONMENT

Once again, our targets are cylinders whose height, as far as the robot is concerned, is infinite. The LIDAR detector is not directly above the arm, so an offset vector needs to be in place in order for the arm to go to the right point on the XY plane. The value of this vector and the vectors that defined the arm had to be adjusted every now and again, since our measurements were far from perfect.

We positioned the target in front of the robot, and then told the arm to look for it. Once it found it and grasped it, we instructed it to set it aside. We tried approaching the target from different angles with varying degrees of success.

Friction isn't taken into account in our program, and in our early tests, the target had to be suspended above the ground in order for the arm to successfully grasp it. Fig. 11 shows one of our later tests. The target is in front of the robot, and supported on a makeshift platform. The arm, when it moves the target to another location will simply drop it to another platform.

Fig. 11 Testing the Arm

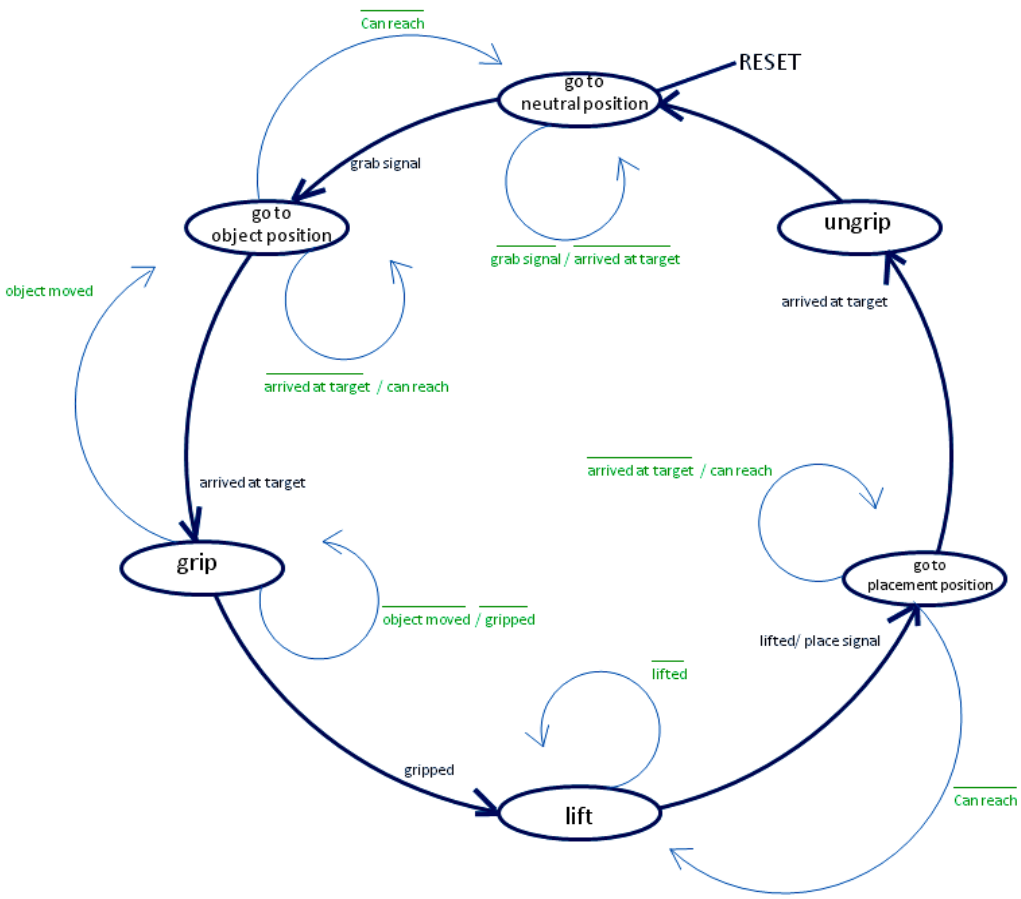

Fig. 12 State Machine for Arm

3.5 FINITE STATE MACHINE

Now that we have defined the inverse kinematics programs for the arm and have gathered data on the surroundings of the robot via LIDAR, we can implement the process of autonomously grasping targets. This requires the process of detecting and grabbing objects to be broken down into several stages (six in our case) and implemented in a finite state machine (Fig.12). The conditions needed to get from one stage to the next (or to backtrack to the previous stage) also need to be defined.

Let us go through an iteration of this state machine: The arm remains in a **neutral position** until prompted by the grab signal, which is a set of coordinates within the arm's reach for it to go towards. Given the signal it **heads towards the object**, and follows the object if it is moving, so long as it is in range. Once the arm has satisfactorily centered the grasper around the object it **grips it**. The arm **lifts the object** once it has finished gripping, and moves the object aside to a **placement position** (when prompted by the user in our experiments). Then, the arm **ungrips** the object, dropping it in the new position. Afterwards, the arm returns to its **neutral position**, ready for the next grab signal. All the while, if something goes astray, the state machine will remain in its current state until the issue is resolved or revert to a previous state.

Comment [MW12]: Does this involve locating it in space, and then setting it down? Simply shoving it? Dropping it from a height?

4. DISCUSSION AND CONCLUSIONS

The LIDAR scanner is able to plot surroundings and give us a good point for the arm to head towards. The arm is able to go towards an object and follow the object if it moves. It even grasps the target and sets it aside, assuming ideal conditions.

The principle of using LIDAR to detect objects and designing programs to give useful instructions to a robotic arm to interact with these objects has been successfully proven. However, the inverse kinematics are still in the process of being refined and a better means of calibrating the arm (and the vectors it is represented by in the programs) needs to be found.

Hopefully, future students in this project will plot surroundings in true 3D, and not have to rely on all obstacles being of regular width and shape. Also, using this arm in conjunction with the SLAM programs on the MAGIC 2010 platform would go very far in proving the principle of using an autonomous robotic arm in emergency work.

Fig. 13Arm attached to the MAGIC 2010 UGV

4. RECCOMENDATIONS

When working on robotic arms with a two dimensional model of the robot's surroundings, a four vector model for the arm proved to be absolutely necessary. The grasper had to be parallel to the ground (XY plane) because if it approached the target (a tall cylinder) at an angle, it would tilt the target away from the robot. Since the LIDAR scanner is positioned above the arm, the target appeared to be moving further away from the arm than it actually was, and it told the arm to go even further towards the target. This ultimately tipped the cylindrical target over. This is why we decided to add the fourth vector to constrain the grasper. Also the geometry of the grasper changed when it began to grip the target, so one must be aware of such changes in the arm's geometry when calculating the kinematics of the arm.

Also, working in two dimensions is not as effective as three and demands absolute accuracy on the part of the inverse kinematics program. The gripper must be centered exactly on the target. If one finger of the gripper touches against the target first it will nudge the target, causing the target to move away. This can make the arm go into a positive feedback loop where every time the grasper accidentally rubs against the target the arm will advance further. Working in three dimensions should mitigate this problem.

5. ACKNOWLEDGMENTS

I would like to thank Professor Daniel Lee for contributing his time and energy to undergraduate research and giving me this opportunity to glimpse into the world of academic research.

I owe a great debt of gratitude to James Yang and Justin Yim, two very capable and patient University of Pennsylvania undergraduate students in Professors Lee's lab who answered my questions and helped me clarify our goals. They have set a very high standard for me and my future research endeavors. Their enthusiasm and perseverance is an inspiration.

Aleksandr Kushleyev and Daniel Mellinger have also been very patient in answering my many questions, and I wish them every success in their robotics ventures.

Of course, all of this would have been impossible without the generosity of the National Science Foundation and its ongoing efforts to educate new generations of researchers.

Comment [MW13]: just contribute to, not into. And just glimpse the world, don't glimpse 'into.'

6. REFERENCES

[1] J. Pearce, (2011, August 15) George C. Devol, Inventor of Robot Arm, Dies at 99. [Online]. Available: http://www.nytimes.com/2011/08/16/business/george-devol-developer-of-robot-arm-dies-at-99.html

[2] C-C. Tsai, et al., "Autonomous Task Execution Using Stereo Vision Camera for a Two-Armed Service Robot" in *Proceedings of 2011 International Conference on System Science and Engineering*, Macau, PRC, 2011 pp.149-154

[3] "A White Paper on LIDAR Mapping" white paper, MOSAIC Mapping Systems, May. 2001.

[4] Unknown, (2011, February 16) MAGIC 2010: Super-smart robots wanted for international challenge [Online] Available: http://www.dsto.defence.gov.au/MAGIC2010

[5] J. Butzke, K. Daniilidis, A. Kushleyev, D. Lee, C. Phillips, M. Phillips, "The University of Pennsylvania Magic 2010 Multi-Robot Team". UPenn. PA. 2010.

[6] Robotis Technical Staff, *Dynamixel AX-12*, Robotis, 2006.

[7] Hokuyo, "Scanning Laser Range Finder UTM-30LX/LN Specification", May, 2009

7. APPENDIX – USING MATLAB TO CONTROL DYNAMIXEL SERVOS

Below is a good source of open source MATLAB code that can be used to control Dynamixel motors:

http://agaverobotics.com/products/servos/robotis/ax12/ax12-open-source-code.aspx

I would like to call the reader's attention to SerialLed which allows the user to directly manipulate the packet of information sent to Dynamixel servos. The data sheets of Dynamixel motors are the best source for learning the language used by their microcontrollers. A *USB2Dynamixel* converter is needed in order to interface a Dynamixel servo to your computer and is available from ROBOTIS.

Below is an example of a code based on SerialLed which is designed to let the user change the ID of a servo so that is can properly respond to commands. The manufacturer's default ID is 1, unless the servo is marked otherwise.

`setCheckSum` is a piece of code written by my colleague James Yang which simplifies the process of making a CHECKSUM which is the Dynamixel's way of preventing error's in message transmission.

```
function Change_ID()

% Oftentimes you will have to change 'COM4' to some other
serial port that is available on your computer. The default
'BaudRate' for a Dynamixel is 1 megabit/second but can be
changed.

    % Set the port paramenter
    s=serial('COM4', 'BaudRate', 1000000, 'Parity', 'none',
'DataBits', 8, 'StopBits', 1);

    % open the port
    fopen(s);

    % display the com port resources
    com = instrfind;
    disp(com);

    id = input(What is the original ID of the servo you
wish to change?: ', 's');
    ID=str2num(id);
    newid = input('What do you want to change the ID to?:
', 's');
    NEWID=str2num(newid);
    % CODE CONTINUES ON NEXT PAGE
```

```
    %---{Data packet, instructing servo to change its ID }-
%[FF, FF, "Original ID", “Length of packet (including
CHECKSUM)”,   “Writing”, “Position in memory of parameter
to be changed (in this case, 3)”, "New ID" ]

    a = [255, 255, ID, 4, 3, 3, NEWID];
    %-------------------------------------------------
    % Set check sum to prevent errors,
    a=setCheckSum(a);

    % display the values in a
    disp(a);

    % binary write
    fwrite(s, a);

    % Expecting a 6 byte status packet
    out=fread(s, 6);

    % Display status packet
    disp(out);

%Run Test which turns on pilot LED of servo to ensure if ID
was set properly.
    b = [255, 255, NEWID, 4, 3, 25, 01];
    %-------------------------------------------------
    b=setCheckSum(b);

    % display the values in a
    disp(b)

    % binary write
    fwrite(s, b);
    % Expecting a 6 byte status packet
    out=fread(s, 6);

    % Clean up
    fclose(s);
    delete(s);
    clear s;
end
```