


Algorithm for Accurate Control of Low-Cost Joint-Based Robots

A PROJECT IN INDUSTRIAL ENGINEERING AND COMPUTER SCIENCE

Massachusetts Middle School Science & Engineering Fair 2017

Siroun Johnson

Amigos K-8 School
Cambridge, MA



Accurate Control of Joint-Based Robots

Table of Contents

Introduction	3
Robotic Arms	3
The Field of Robotics	3
Joints	4
Optimization	4
Hypothesis	6
Goals	7
Materials	8
Procedure	9
Changes Made	10
Pilot Testing Process	13
Field Testing Process	15
<i>depth</i>	16
<i>delta_d</i>	16
<i>pick_up</i>	16
<i>forward</i>	17
<i>s</i>	17
<i>delta_s</i>	18
<i>angle</i>	18
<i>dx</i>	18

Accurate Control of Joint-Based Robots

Results	19
Python Algorithm	19
Important Variables to Optimize	19
Quantitative Grading of Drawings	20
Results of Optimization	21
How the Code Works	22
__init__	22
wait	22
get_speed	23
home	23
distance	23
get_time	23
get_depth	23
move_to_point	24
draw	24
Next Steps	25
Conclusions	26
Acknowledgments	27
References	28

Accurate Control of Joint-Based Robots

“Though further developments might enable people with tetraplegia to achieve rapid, dexterous actions under neural control, at present, for people who have no or limited volitional movement of their own arm, even the basic reach and grasp actions demonstrated here could be substantially liberating, restoring the ability to eat and drink independently.”

Hochberg et al. 2012 in Nature

Introduction

Robotic Arms

Robotic arms are very expensive. Computer-integrated manufacturing systems have been used in many industries for more than four decades (Taylor et al. 2009). Medical robotic arms that aid in surgery use pneumatic actuators, meaning that their actions are controlled by compressed air. Human-friendly robots use joint-based technology, where the joints, also called DOFs are controlled by servos (Jeong et al. 2001). For example, a human arm exoskeleton requires seven joints, so it is 7 DOF. Computer-integrated surgery uses robot technologies (Taylor et al. 2009). Robotic arms are also made for other purposes, such as helping people with tetraplegia, which is a type of paralysis caused by spinal cord injury, brainstem stroke, or other ways of disconnecting the brain from the body. Robotic arms have been designed for people with tetraplegia, giving these people the ability to reach and grasp (Hochberg et al. 2012). Robotic arms that are being made to aid with tetraplegia are not controlled by compressed air. However, controlling movements precisely is relevant to these robotic arms as well. Any kind of robotic arm must have very highly controlled movements.

The Field of Robotics

There are entire research journals that publish work about improving the accuracy and repeatability of robot motion. For example, the *International Journal of Robotics Research* has been publishing research for two decades on quality, cost, and studies of how to improve quality at lower costs. According to the *Springer Handbook of Robotics* (Siciliano, Bruno, and Oussama Khatib 2016), the foundations of the field of robotics are kinematics, dynamics, mechanisms and actuation, sensing and actuation, motion planning, motion control, and force control. Kinematics is about figuring out how to describe the location of a robot using coordinates in space. Dynamics is about describing the motion of a robot. Mechanisms and actuation is about how to make a robot move. Sensing and actuation is about detecting where a robot is and how to move in response to this information. Motion planning is how to generate

Accurate Control of Joint-Based Robots

a trajectory of motion. Motion control is about communicating information from the software to the hardware of the robot. Force control is about making sure that the robot does not violate any constraints. For example, if the robot is supposed to hold an egg, it should only apply a certain amount of force so as to not break the egg. All of these are involved in building a robotic arm.

Joints

Controlling a robotic arm means controlling its joints. In order to control a joint, it is necessary to command a joint to move by a particular angle. The angle is determined from comparing where the robot is currently to where the robot should be at the end of the movement. Calculating the angle requires inverse kinematics. To plan the dynamics of the robotic arm motion, variables have to be defined. Certain lengths are important to specify, and an endpoint is necessary as well. The motion of each joint must be individually specified and controlled. Specifying this depends on how precisely the servos can be controlled. Inexpensive servos, such as the ones used in joints in this project, are less precise than more expensive ones. Making the joints move requires writing a computer algorithm that controls the servos. The algorithm tells the servos how many degrees to turn and what direction. The degrees are calculated from the inverse kinematics. A subroutine in the algorithm does the inverse kinematic calculations. Inexpensive servos do not provide much sensing capability, but some errors can be programmed, such as detecting when invalid information is sent to the servo. Motion planning involves lots of trial and error in defining variables and testing them to find out which variables are useful and what are the best combinations. Controlling the motion of the servos meant issuing commands from the computer to the Pololu to the servos. To check that it goes through, it is necessary to run tests to obtain information from the servos back through the same route. Therefore, it is necessary to code a subroutine that gets all the positions of the servos. In a robotic arm, force control is mostly about how fast the servos are moved. Variables can be defined to control these.

Optimization

A single motor in a pneumatic-controlled robot can cost between \$2,000 and \$5,000, and there

Accurate Control of Joint-Based Robots

are several motors per robot, making the total expense of industrial and medical robots very high. This makes them very accurate, but the price is prohibitive. Accuracy and repeatability are the two most important features of quality in a robot. This project presents an innovation to program an algorithm to control an inexpensive robot very accurately with repeatable outcomes. The joints in the robot that was designed for this project are controlled by servos, which cost about \$15 each. The entire robotic arm that was built for this project costs around \$130. The problem investigated was one of industrial engineering, using computer science to optimize the design. The optimization involved improving how well the robotic arm could draw, so that there would be a way of measuring accuracy. To do this, it was necessary to select the most impactful variables and test them in various combinations (24 combinations of variable values in the first phase of testing, and 80 combinations in the second phase). Then, code was created to grade pictures taken of images drawn by the robot at each combination of variable values. Once all the pictures were graded using that code, the gradings were used to find the optimal variable values.

Accurate Control of Joint-Based Robots

Hypothesis

It is possible to design an accurate and simple control system for an inexpensive 4 DOF (four degrees of freedom) robotic arm. This requires:

- ❖ Designing and building an inexpensive robot with four servos
- ❖ Specifying the inverse kinematics for robotic arm control
- ❖ Developing code to control and coordinate the joints
- ❖ Experimenting with different control parameters that can be optimized
- ❖ Creating algorithms to compare accuracy
- ❖ Identifying the most accurate combinations of parameters
- ❖ Interpreting why those combinations are most accurate
- ❖ Explaining how the code works to control the robot

Accurate Control of Joint-Based Robots

"It is strange that only extraordinary men make the discoveries, which later appear so easy and simple."

*Georg C.
Lichtenberg
(1742-1799)*

Goals

- ❖ One way to improve future robot designs is to make accurate robots inexpensively. An inexpensive robot has to first be designed and built before code can be optimized to control how it works.
- ❖ Drawing is a very detailed and complex process. Drawing pictures is used so that accuracy can be measured, in order to solve the problem of having accurate control and repeatable outcomes. Another goal is to create a simple algorithm to draw detailed pictures.
- ❖ Once the algorithm functions, it can be optimized to produce accurate images with repeatable outcomes. To obtain these images, it is necessary to design a method for measuring accuracy.

Accurate Control of Joint-Based Robots

Materials

One of the goals of this project is to have the total cost relatively low. In the list below, **the total cost is \$124.25.**

Item	Robotic arm materials	Quantity	Price (\$)
1	Polulu mini maestro 18-channel USB servo controller	1	\$39.95
2	USB 2.0 cable - A-male to mini-B - 3 feet	1	\$4.20
3	VIMVIP 30cm male-to-female servo extension lead wires (12-pack)	1	\$8.59
4	DF Metal Geared 15kg standard servo 270°	3	\$43.50
5	Goteck Micro metal gear servo	1	\$6.90
6	AA batteries	4	\$3.77
7	Tactic 4-cell AA battery holder with Fut-J connector	1	\$3.88
8	McMaster cast acrylic sheet (plastic for 2D laser printer)	1	\$13.46

Accurate Control of Joint-Based Robots

Procedure

Preliminary Research

1. Review designs of robotic arms.
2. Decide on configuration for robotic arm.
3. Solve forward and inverse kinematic equations for controlling servos with code.
4. Create the physical structure of the robotic arm from the materials.
5. Obtain basic Python subroutines for communicating with servo controllers, and also for converting images to points.

Experimental Phase

1. Write an initial outline for Python code that will make the robotic arm draw.
2. Get the first algorithm to work, with a lot of debugging!
3. Select a complex SVG image.
4. Design measures of accuracy and detail.
5. Invent and tweak various parameters in the algorithm to find a combination that works best.
6. Continue to improve overall algorithm.

Accurate Control of Joint-Based Robots

Changes Made

DATE OF EXPERIMENT	FOCUS OF CHANGE	OUTCOME
1/20/2017	<i>depth</i> parameter definition	While making the eyes of a smiley face (for practice), it was difficult to draw. A global depth parameter was defined to consolidate coding for a z-coordinate so that it could be used by a variety of functions. Depth controls the z-coordinate and therefore how faint or dark the line is. For now, determined that: <i>Depth</i> = -7.45 is the optimum.
1/25/2017	Timing	Making the robot move faster made the lines more accurate by being straighter.
1/27/2017	Organization	Changing the functions into a class (collection of functions) made the program easier to use and more organized.
1/27/2017	Protecting from self-destruction	There is a region that, if the robot arm is in contact with it, some of the servos will burn out, so it was necessary to determine where to draw the image so the robot does not go into that region. 1 inch = 96 pixels 2.45 cm = 1 inch So, 1 cm = $96/2.54$ pixels = ~38 pixels
1/28/2017	Efficiency	Previously, to draw two lines, it was necessary to have 13 lines of code. This change involved using the code that changed an SVG file into a collection of points, and the function called <code>move_to_point</code> , together to create the drawing. Now, there is one function that puts this all together in a FOR loop and some IF clauses.
1/29/2017	Picking up the pencil	NaN = not a number Whenever it is necessary to pick up the pencil, it is represented in the collection of points as a NaN. If the program gets to a NaN, it should go up, go over, and then go down. This change enabled picking up the pencil whenever the input was a NaN.

Accurate Control of Joint-Based Robots

2/3/2017	Optimizing the drawing speed	The author drew a 10 cm line, and it took 1.26 seconds. So the speed was 7.9 cm/s. The robot was then programmed to draw at the same speed.
2/9/2017	Measuring accuracy	Up until now, accuracy was determined by looking at the drawing to see if it seemed more accurate. A simple square was converted to SVG so that it would be possible to compare the quality of drawing a right angle.
2/9/2017	Optimizing the depth coordinate	Determined that having more than one paper underneath the robot, it is necessary to change the <i>depth</i> parameter.
2/9/2017	Neater drawing	When the screws on the robot are tightened, the drawing is less messy.
2/10/2017	Mirror image drawing	It was discovered that the drawing was being made as a mirror image, so the y coordinate was switched to the opposite sign by multiplying it by -1 to fix this. Now the robot can write!
2/11/2017	Benefits and tradeoffs of parameters	<p><i>depth</i></p> <ul style="list-style-type: none"> ↑ = higher, lighter, might not draw ↓ = lower, darker, might break <p><i>pick_up</i></p> <ul style="list-style-type: none"> ↑ = go higher, make sure it doesn't hit paper, can be inefficient ↓ = lower, can be faster, can accidentally draw <p><i>forward</i></p> <ul style="list-style-type: none"> ↑ = farther away, can be shaky, less accurate but if it's too high then the range of function is reduced ↓ = closer, can run into itself, range of motion is lower but if it's too low then the range of motion is less <p><i>s</i></p> <ul style="list-style-type: none"> ↑ = faster, less accurate, good for curves, but slams down on the paper if too fast ↓ = slower, better for straight lines <p><i>dx</i></p> <ul style="list-style-type: none"> ↑ = less accurate, less points, less coordination, faster ↓ = more accurate, more points, more coordination, but takes longer the smaller it is
2/12/2017	Optimizing range of motion	A function was written to check if distance from tip of pencil to origin is greater than a specified value. If the distance is close, then the pencil is angled toward the robot base (225°), and if it is farther, the pencil is

Accurate Control of Joint-Based Robots

		<p>angled away from the base (135°). This would give more range of motion. A problem was discovered: the servo controlling the angle that the pencil contacts the paper has a maximum of 180°, so this modification is not possible.</p>
2/12/2017	Smoothness of line	<p>The farther away the tip of the pencil is from the robot base, the faster the robot can draw. The closer, the slower it draws. h = length of the paper perpendicular from the base of the robot s = drawing speed $forward$ = amount that paper is moved forward from the base of the robot $max\ s$ = maximum speed $min\ s$ = minimum speed $s = \frac{max\ s - min\ s}{h} (x - forward) + min\ s$ The optimum speed is given by s in the equation.</p>
2/14/2017	Optimizing the angle of the pencil	<p>Determined how to calculate the distance from the origin to a point in 3 dimensions, using the initial and final x, y, and z coordinates of two points (this is the Pythagorean theorem in 3 dimensions), in this case, $x_2 = y_2 = z_2 = 0$:</p> $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$ <p>So, the simplified version is:</p> $d = \sqrt{(x_1)^2 + (y_1)^2 + (z_1)^2}$ <p>Once this distance is determined, the robot can be told to orient the pencil at a particular angle that depends on the distance.</p>

Pilot Testing Process

The optimal values for the parameters were determined using multiple tests. After building the set of variables, two figures (converted SVG files) were selected based on diverse properties. One was a square (with straight lines) and the other was a cat (with mostly curved lines). Values of the parameters that were estimated as making reasonable drawings were established as the middle values, and then one value less and one value more than this were chosen for each variable, so that three values could be tested for each variable. The intervals were not always equal because the goal was to test the entire range of values that could work. Then, for each figure, drawings were made, using each of the three values of a parameter. These were labeled A, B and C. Four people then rated each of the three drawings from best (1) to worst (3), and these ratings were averaged.

Optimization of the variables was calculated by a weighted average so that the robot would be accurate with both straight lines and curves (Phase 10). Based on examination of a variety of drawings, it was determined that most drawings have 90% curves and 10% lines, thus these weightings were used to determine the optimal values.

An example of how averages were determined is shown in the table below.

<i>depth</i>	<i>line rating</i>	<i>curve rating</i>
-7.2	2.75	1.75
-7.5	1.75	1.25
-8.0	1.50	3.00

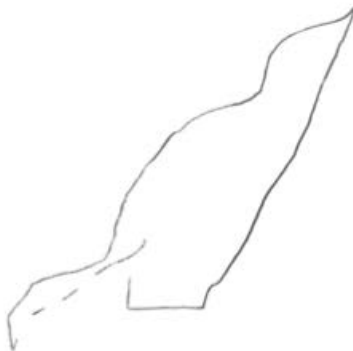
Accurate Control of Joint-Based Robots

The pilot testing phase provided the following initial set of optimal variable values.

Variable	Optimal Value
depth	-7.5
delta_d	1.0
pick_up	1.45
forward	10

Variable	Optimal Value
s	15
delta_s	2.5
angle	160
dx	3.0

Below are shown drawings made by the robot, with non-optimal and optimal values of the variables. These are compared to the original SVG image input to the program.



Early values of
parameters



Optimal values of
parameters

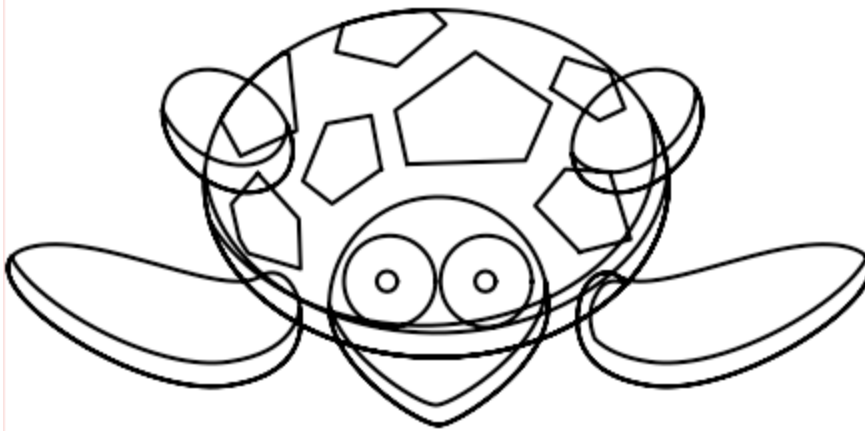


SVG of cat that was
input

Field Testing Process

Following the pilot test, a larger and more quantitative field test process was designed to better determine the optimal values for each variable. The variables were tested one at a time. Ten values of each variable were tested, using the optimal value from the pilot testing, and creating a set of ten values that ranged above and below it. A complex image of a turtle was used which included detailed lines, so that grading could include a wide range of possibilities from very poor (low grade) to very good (high grade). After each drawing was made by the robot, a photograph of the drawing was captured and compared to the original image that was input to the code. The images were all converted to SVG before comparing them.

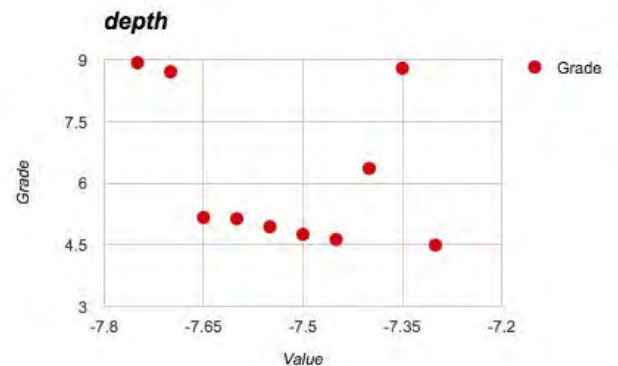
Turtle Image Used for Testing



Accurate Control of Joint-Based Robots

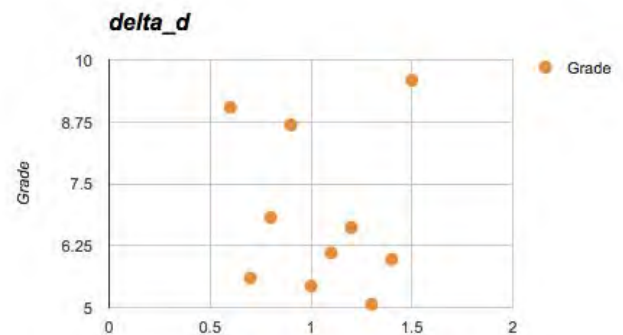
depth

This variable controls how high up from the paper the pencil is, in other words, the z-coordinate with the exception of when the pencil is being lifted off the paper. It is a negative number because it is below the center of the servo that is used as the origin. The ***depth*** variable has two values that were very similarly optimal. These are -7.75 and -7.35. These two values must be tested again in the final round to determine which produces the best grade.



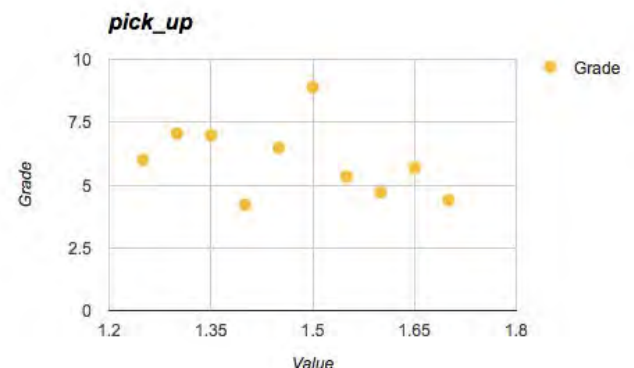
delta_d

This variable is the amount added or subtracted from *depth* to make the z coordinate proportional to the y coordinate. So, the pencil would be lower when farther from the origin and higher when closer. The optimal value of ***delta_d*** is 1.5.



pick_up

This is the amount, in centimeters that is added to the *depth* variable when picking up the pencil. The optimal value of ***pick_up*** is 1.5.



Accurate Control of Joint-Based Robots

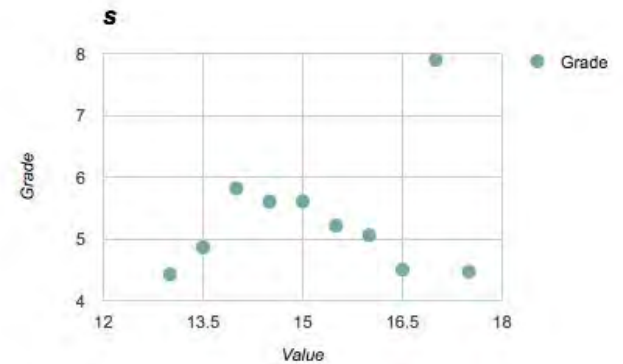
forward

This is the amount, in centimeters, that shifts the paper up. If this were 0, the robot might get a point too close to the origin and break. The optimal value of ***forward*** is 8.5.



s

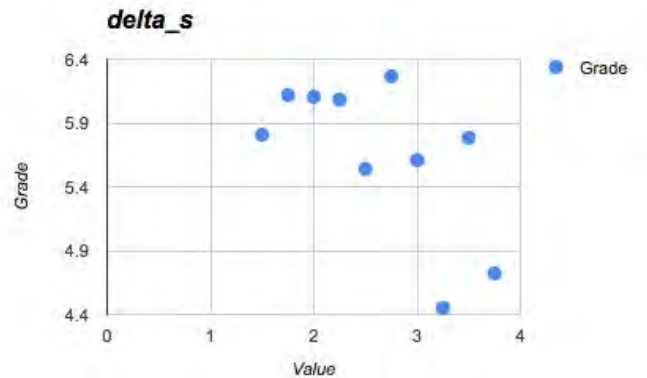
This variable is the target speed of the robot in centimeters per second. The prediction was that when ***s*** is larger, the robot draws faster but it is less accurate. This is good for curves, but going too fast, the robot arm slams down on the paper. Going slower is better for lines. The optimal value for ***s*** appears to be 17.0, but this might be an anomalous value. Perhaps 14.0 is the optimal value. This will need to be tested.



Accurate Control of Joint-Based Robots

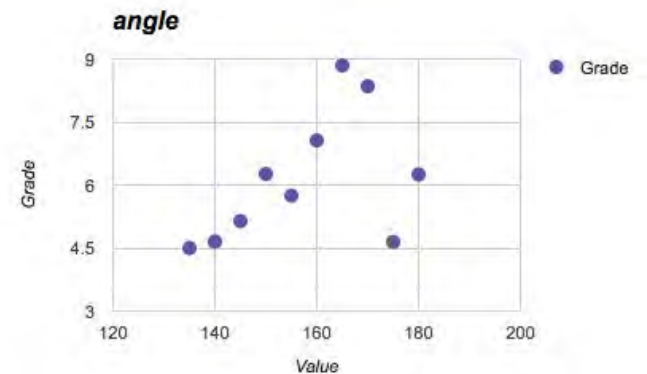
delta_s

This variable is the amount added or subtracted from ***speed*** to make the speed proportional to the y coordinate. So, the pencil draws faster when farther away from the origin and slower when closer. The optimal value for ***delta_s*** is 2.75.



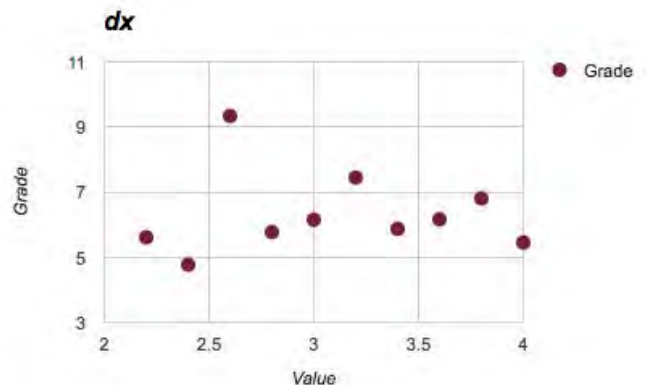
angle

Angle controls the angle at which the pencil touches the paper. The optimal value for ***angle*** is 165.



dx

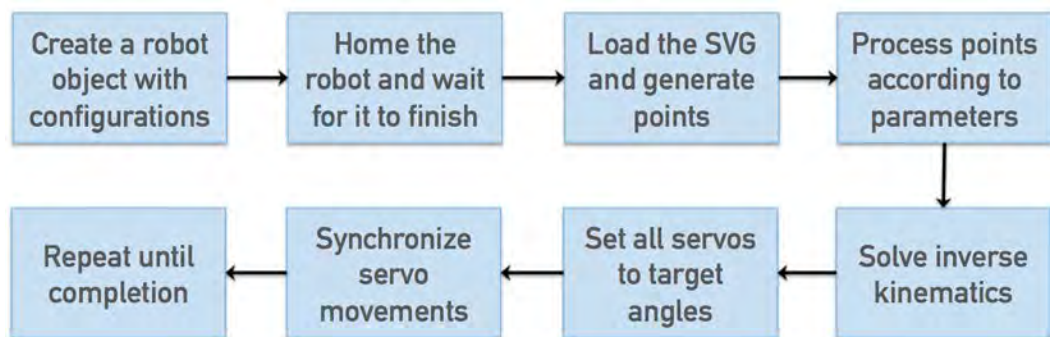
This variable determines how many points the robot is told to draw, in points per centimeter. The optimal value for ***dx*** is 2.6.



Accurate Control of Joint-Based Robots

Results

Python Algorithm



An algorithm was created using Python to make the robot draw. The flowchart of the algorithm is represented above. Each box is a complex step in the robot control. These pieces encompass all of the major foundations of robotics: kinematics, dynamics, mechanisms and actuation, sensing and actuation, motion planning, motion control, and force control (Siciliano, Bruno, and Oussama Khatib 2016).

Important Variables to Optimize

There are 8 variables that are essential to controlling the robotic arm. One variable (***delta_a***) was determined not to be useful. The variables that control the coarse mechanics are ***depth***, ***pick_up***, ***forward***, ***s***, ***angle***, and ***dx***. The variables ***delta_d*** and ***delta_s*** make depth and s proportional to the y coordinate. They are not as essential, because the robot movement would still be able to draw without these variables, but these two variables refine the movements.

The optimal values of the variables were determined through two phases of testing, summarized below. Final testing of the robot was carried out with the field test optimal values of the variables.

Accurate Control of Joint-Based Robots

The two testing phases provided the following sets of optimal variable values.

Variable	Pilot Test	Field Test
<i>depth</i>	-7.5	-7.75 or -7.35
<i>delta_d</i>	1.0	1.5
<i>pick_up</i>	1.45	1.5
<i>forward</i>	10	8.5

Variable	Pilot Test	Field Test
<i>s</i>	15	17 or 14
<i>delta_s</i>	2.5	2.75
<i>angle</i>	160	165
<i>dx</i>	3.0	2.6

The four combinations were then tested, and the best combination is *depth* = -7.75 and *s* = 17.

Quantitative Grading of Drawings

The pilot process used human perspective to grade the drawings. The field test process improved upon this by creating a quantitative measure of how good the robot's drawings were compared to the original drawing input to the algorithm. Code was adapted from Doxygen (2017). There were several steps in grading an image. The inputs are the SVG that the drawing was made after, and a picture of the drawing made by the robotic arm. First, before any images are input to compare the initial image to, several reference points are plotted on the SVG image. The quantity of these points is *a*. Then, once the drawing is input, the grading algorithm connects several points to the initial image. The amount of these connections is *b*. Some of these connections will not be completely correct. Next, of these connections, the computer finds which connections are true connections. The true connections are *c*. The total grade is calculated with these points weighted:

$$5\% \Rightarrow \frac{b}{a}$$

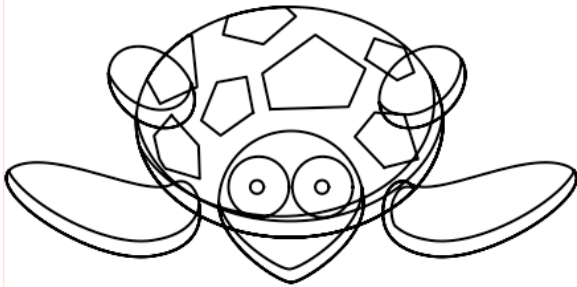
$$95\% \Rightarrow \frac{c}{b}$$

The highest possible grade is 100 and the lowest is 0. If the drawing is too light and difficult to see, the grader automatically gives it 0 for its grade.

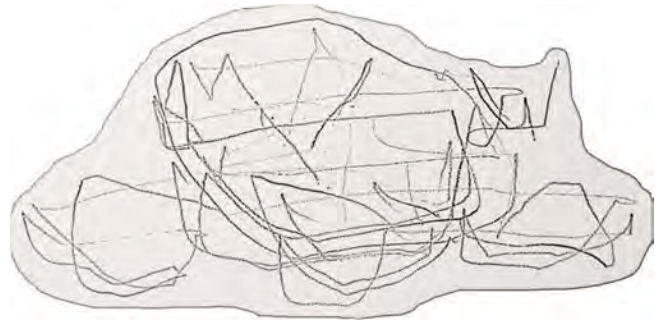
Accurate Control of Joint-Based Robots

Results of Optimization

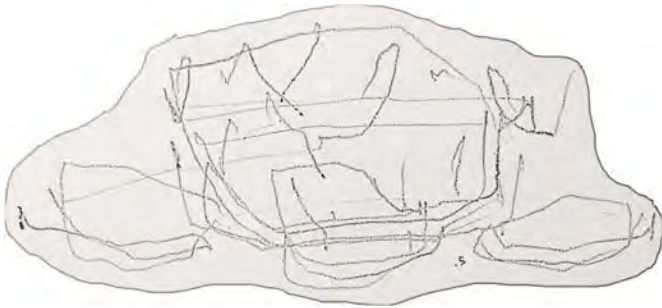
Below are shown drawings of the turtle with three sets of parameters: non-optimal, pilot test values, and field test values.



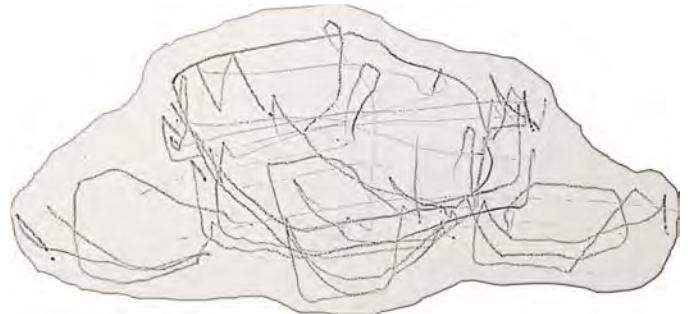
Original SVG image of turtle



Turtle drawn with non-optimal values



Turtle drawn with pilot test values



Turtle drawn with field test values

	<i>depth</i>	<i>delta_d</i>	<i>pick_up</i>	<i>forward</i>	<i>s</i>	<i>delta_s</i>	<i>angle</i>	<i>dx</i>
Non-optimal	-7.5	2.0	1.45	10	15	2.5	180	5.0
Pilot test	-7.5	1.0	1.45	10	15	2.5	160	3.0
Field test	-7.75	1.5	1.5	8.5	17	2.75	165	2.6

Accurate Control of Joint-Based Robots

"Beauty is the summation of parts working together in such a way that nothing needs to be added, taken away, or altered."

Carlotti, Italian Painter

How the Code Works

The algorithm created to control the robotic arm has several pieces of code, each with its own specific purpose. For example, one code might control the servos while another calculates the inverse kinematics. Then, these pieces are all imported into one algorithm, called *Robot*. The code has several parts that work together, called functions, that are detailed below.

`__init__`

This is a part of the code that takes all the inputs and allows the use of them in the other functions. Here are the inputs and their meanings:

- *servos*: a list of all the servos
- *depth*: the z-coordinate in centimeters
- *delta_d*: the amount added or subtracted from depth to make the z coordinate proportional to the y coordinate
- *pick_up*: the cm added to depth when picking up the pencil
- *forward*: the amount in centimeters that viewport moves up
- *speed*: the speed at which the robot draws in cm/sec
- *delta_s*: the amount added or subtracted from speed to make the speed proportional to the y coordinate
- *angle*: the angle at which the robot draws in degrees
- *lengths*: the four lengths of the robot in centimeters
- *viewport*: the size of the paper that the robot is drawing on

`wait`

This function allows the robot to wait until it is done moving before it goes to another spot, rather than go to every spot as fast as the robot possibly can because it is constantly being fed commands to go to a certain point. This would most likely burn out the servos.

Accurate Control of Joint-Based Robots

get_speed

This sets the speed proportional to how far away the point of the pencil is from the origin, which is the middle of the robot. If the tip of the pencil is far away, the robot will go faster by $s + \text{delta_s}$, but if it is closer, it will go at a speed of $s - \text{delta_s}$. This function is not absolutely necessary for drawing, but it greatly improves the quality of the drawing.

home

This function homes the robot, meaning that all of the robot's angles are set to 0. This is used just before the drawing process, just to calibrate all the servos.

distance

This function has no direct purpose except to calculate the distance between two points in 3D for the use of other functions such as *get_time*. This is the formula to find the distance between two points with coordinates (x_1, y_1, z_1) and (x_2, y_2, z_2) :

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

get_time

This uses the equation, $\text{distance} = \text{speed} * \text{time}$, to solve for the amount of time it will take to get from one point to another. So, this takes in two points and the speed at which the robot should be going, and then finds the amount of time it will take.

get_depth

This sets the depth, or the z-coordinate, which becomes proportional to the distance from the robot in this code, using *delta_d* and *depth*. If the tip of the robot is far away, the robot will go higher by $\text{depth} + \text{delta_d}$, but if it is closer, it will go lower, $\text{depth} - \text{delta_d}$. This function is not absolutely necessary for drawing, but it greatly improves the quality of the drawing.

move_to_point

This is one of the two most important functions. This function takes in the coordinates of one

Accurate Control of Joint-Based Robots

point as (x, y, z) and also ϕ , which is the angle at which the pencil touches the paper. First, the computer uses *get_time* to find the amount of time it will take to move the tip of the pencil from the current point to the point that was input. The computer already knows what the current point is because every time that the computer tells the servos to move, it resets the current point, including when using the *home* function. Next, the computer uses *solve_ik*, a class in another piece of code, to solve the inverse kinematics, or the combination of angles that would move the robot to a desired point, in this case the (x, y, z) that was input. There are 4 possible solutions, but the computer chooses the first one because the other solutions would be options that the servos simply cannot do, for example turning around and drawing from below. Next, the code sets the target for all of the servos, using the angles calculated earlier and uses the already computed time to bring them all there at the same time, to be efficient, but also not leave any unnecessary marks on the paper. Finally, the code resets the current point.

draw

The other important function is *draw*, which uses the points made from the SVG image input and *go_to_point* to tell the robotic arm where to go. First, the computer uses a piece of code that was imported into this one, called *Drawing* to take the SVG image and turn it into points. The way *Drawing* works is it turns the image into angles, then lines, and finally a certain amount of points depending on dx (points/cm). The image has to be SVG (Scalable Vector Graphics) because it is made out of lines, which are easy to identify. When the robot has to pick up the pencil and move to another line, the code puts in a NaN (not a number). After finding the points, the robot goes through a for loop in which every one of those points goes through the *move_to_point* function and gets drawn, every time resetting the current point. When the for loop gets to a NaN, it picks the arm up by *pick_up* and moves to the next point in the list. Once it gets through the whole loop, the robot has made a drawing that will hopefully look like the SVG input!

Accurate Control of Joint-Based Robots

Next Steps

In the future, the robot can be improved by using two servos for the top joint to improve accuracy. In addition, there can be shorter extension wires because the wires often get in the way of the drawing process and decrease drawing quality.

More granular and multidimensional testing could be performed around the current optimal parameter values to make the drawings even more accurate.

- For example, a variable could be held at a certain value that was optimal in a test, and values of a second variable could be tested. Then the first variable could be increased or decreased slightly, and the second variable tests could be run again to find out if there is a better combination of the two variables than testing the variables separately.
- Code could be developed to randomly select some combinations of variables, to try to figure out other combinations to test. There are six variables that are all important, which are ***depth***, ***pick_up***, ***forward***, ***s***, ***angle***, and ***dx***. Instead of testing a variable and then setting it to its optimum value before testing the next variable, the three best values of the first variable could be run with all of the values of the second one to see if it makes a difference. Also, a different order of testing the variables could be tried.

The servos used in this robotic arm had a precision level of 1 degree. This limited the accuracy because the robot could not be made to draw more precisely. More precise servos would increase accuracy, but they also increase cost. A next step could be to test more precise servos and then do a cost-benefit analysis.

Stronger material for the robot would improve accuracy because the robot would be less wobbly, but would weigh more. But, if something were to break, it would be more costly to repair the damage. Again, a cost-benefit analysis could be done if a second robot was built from stronger material.

Conclusions

In his Foreword to the *Springer Handbook of Robotics*, Bernard Roth, who is a Professor of Mechanical Engineering at Stanford University, says that there is a “modern trend of carefully detailed mechanical and electronic design, optimized software, and complete system integration” that has become the “norm to this day” and that “this combination represents the hallmark of most highly regarded robotic devices” (Siciliano, Bruno, and Oussama Khatib, 2016). Software is used for control of motion and control of interaction forces between the robot and its environment.

It is possible to build an inexpensive robot whose accuracy can be improved. However, this cost effectiveness creates a lot of barriers because low-cost servos are highly inaccurate.

There is probably a trade off between better code and more precise servos. A lot of research literature focuses on improving accuracy by innovations in hardware. This project has shown that accurate control of joint-based robots can also be optimized by improving software. Specifically, variables can be introduced, and then optimized, which will improve the robot's accuracy.

Acknowledgments

The author wishes to thank the following people who have supported her during the process of doing this project.



Lujing Cen

Mentor

Computer Science Major
Massachusetts Institute of
Technology



Laurie Ferhani

Middle School Science
Teacher

Amigos K-8 School
Cambridge, MA



Hannah Sevian

Mother

Chemistry Professor
University of Massachusetts
Boston

References

1. Clothier, Kurt E., and Ying Shang. "A geometric approach for robotic arm kinematics with hardware design, electrical design, and implementation." *Journal of Robotics* 2010 (2010).
2. Doxygen. "Feature Matching." *OpenCV: Feature Matching*. Doxygen, n.d. Web. 28 May 2017.
3. Edan, Yael, et al. "A three-dimensional statistical framework for performance measurement of robotic systems." *Robotics and Computer-Integrated Manufacturing* 14.4 (1998): 307-315.
4. Hochberg, Leigh R., et al. "Reach and grasp by people with tetraplegia using a neurally controlled robotic arm." *Nature* 485.7398 (2012): 372-375.
5. Jeong, Younkoo, et al. "A 7 DOF wearable robotic arm using pneumatic actuators." *Proceedings of the 32nd ISR (International Symposium on Robotics)*. Vol. 19. 2001.
6. Kostic, Dragan, et al. "Modeling and identification for high-performance robot control: An RRR-robotic arm case study." *IEEE Transactions on Control Systems Technology* 12.6 (2004): 904-919.
7. Siciliano, Bruno, and Oussama Khatib, eds. *Springer handbook of robotics*. Springer, 2016.
8. Taylor, Russell, et al. "A steady-hand robotic system for microsurgical augmentation." *The International Journal of Robotics Research* 18.12 (1999): 1201-1210.