INTRO TO ROBOTICS JOURNAL
COMP 20170
GROUP 4:

TEAM MEMBERS:

- Mynah Bhattacharyya
- Luke Lynam
- Adam Lacey
- Jon Eckerth
- Ben McDowell

Assignment 3: Robot Arm: Steps

- Sweep left and right until an object is found, pick it up and place it at specified start position

General Roles:
Adam - ROBOTC code, build and test
Mynah - Project manager, ROBOTC code, build, journal
Jon - ROBOTC code, build and test, rotational matrices calculations
Ben - rotational matrices calculations, ROBOTC code
Luke - EV3 Build, journal, ROBOTC code

ASSISTIVE LINKS USED FOR PROJECT:

EV3 Robotic Arm: WITH CODE
Robot arm - LEGO 45544 MINDSTORMS Education EV3 Core Set
"The LEGO Mindstorms EV3 Robot Arm The Cables & The Program"
https://le-www-live-s.legocdn.com/sc/media/files/ev3-program-descriptions/ev3-program-description-robotarm-621fa90f70727e1fa8b5e82a9bdb3446.pdf
https://robotics-explained.com/transformation

Gantt Chart (Note: IP = In Progress)

Tasks:	Status	Person	Date Finished
EV3 build robot arm following booklet	Complete	Everyone	09/3
Project familiarisation, first code pass	Complete	Mynah	20/3
Write function to rotate base joint	Complete	Mynah, Adam	23/3
Write function to rotate arm joint	Complete	Mynah, Adam	23/3
Write function to rotate clamp motor to clamp	Complete	Mynah, Adam	23/3
Test motion of clamp (unique motion)	Complete	Adam, Luke	23/3
Test colour sensor response to red object (does it stop the rotation?)	Complete	Ben, Luke	24/3
Test motion, debug code	Complete	Luke, Adam, Jon	24/3
Find rotational matrices of each step of movement (based on team-specified angles)	Complete	Jon	25/3
Final tweaks	Complete	Luke, Adam, Jon	24/3
Shoot Video	Complete	Luke, Ben, Jon, Adam	24/3
Code cleanup	Complete	Mynah	25/3
Journal	IP	Mynah, Luke	25/3

DAILY LOG 1:
Mynah, Thurs 09/3

After we finished our video for Assignment 2, we began to work on our build for the Robot Arm by taking apart our previous build and closely following the instructions in the Lego build booklet on Brightspace. Since we began towards the end of class due to unforeseen complications with the previous project, we had to continue past the class time to complete the build - everyone except Ben stayed to assist with the build. Adam and Luke worked on the main build from start to end, while Jon and I would skip ahead in the instructions and put together separate smaller components to be passed to the other team who would piece them together in the main build. Since we had not done the example code review of this project in class, we were not really sure how we would tackle the code or what potential changes might be needed to the build to accommodate the direction we would take. We figured that we would be able to swap out sensors/rebuild parts etc as needed. Adam decided to take the robot home as at the time he was the only one with a working ROBOTC licence (we were not sure when Professor Eleni would be renewing the rest), while Jon kept the box with spare parts. Since we had a 2 week break upcoming, my hope was that we would be able to work on code parts online and send it over to Adam for testing - if any rebuilds were necessary we would hopefully be able to meet up.

At the end of the class, the pragma configs for the sensors were as follows (as per booklet build):

```
#pragma config(Sensor, S1,     touchSensor,    sensorEV3_Touch)
#pragma config(Sensor, S3,     colorSensor,    sensorEV3_Color, modeEV3Color_Color)
#pragma config(Motor,  motorA,          armMotor,      tmotorEV3_Large, PIDControl, encoder)
#pragma config(Motor,  motorB,          leftMotor,     tmotorEV3_Large, PIDControl, driveLeft,
encoder)
#pragma config(Motor,  motorC,          rightMotor,    tmotorEV3_Large, PIDControl,
```

I figured we would be able to move the colour sensor from the arm to the clamp later, as we ran out of time and had to go home.

Total hours worked: 2-3hrs

DAILY LOG 2:
Mynah, Mon 20/3

After reaching past the midpoint of the break, I decided to start working on the project properly. Having previously messaged and contacted other team members over several days, both as a group and individually, and not hearing much of a response back from them, I decided to familiarise myself with how I might go about starting the code on my own in a more procedural way rather than relying on the limited grasp of rotational matrices provided

in class. Since we had not had time to do the example code review of this project in class as well, I watched a few Youtube videos and found some .rfw “block code” examples that briefly explained that the rotation of the arm was based on set times and set motor speeds.

This allowed me to identify 3 possible ways of rotating a joint in its particular plane of movement:

1: Move by motor encoder’s recorded rotation (using setMotorTarget)
2: Move by time and a specified motor speed (using setMotorSpeed and wait commands)
3: Move by gyroscope sensor’s angle rotation (using getSensorValue like in our old code’s turn function from previous assignments)

For the time being, I decided to go with setMotorTarget, since I wasn’t sure if we had the gyro sensors set up on our build, and since motor encoders are independent of any build adjustments. I figured that with adequate modularisation, it would be easy to swap out setMotorTarget with some other usage.

Now my challenge was to tackle the entire project’s movement.
Initially, I tried to break the entire task up into a set of steps/actions that the robot would have to take, which were:

1: Rotate right and left until clamp either “sees” an object or rotation hits 180/-180
2: If object seen, rotate armMotor down
4: Run clampMotor to clamp object
5: Rotate armMotor up
6: Rotate base back to start position
7: Rotate armMotor down
8: Unclamp to drop object
9: Reset robot: lift arm up to base, start program again

Based on this, I could identify 3 main movement functions required:

1: rotateBase(int motorTarget): which would take in a positive or negative angle value to rotate the base joint left or right.
2: liftArm(int motorTarget): which would take in a positive or negative angle value to rotate the mid arm join up or down.
3: moveClamp(int motorTarget): which would take in a positive or negative angle value to clamp onto or unclamp from an object (like opening or closing a fist)

All of these I based on setMotorTarget, and some fixed speeds for each joint as global variables that could be used to adjust how fast or slow we wanted that particular movement, to be tweaked during testing. I figured that if I could get one of these functions to work based on setMotorTarget, the rest would work as well or could be swapped out with ease. I am also working off the assumption that we will move the colour sensor to the clamp, so when the clamp “sees” a red object in its sweep of motion in one angle turn, it can then signal the function to stop.

After this, it was easy to fill out these functions and then add a couple of complex functions that utilised these basic movement functions to delineate “tasks”:

1: doSweepSearch( ): which rotates the base in left and right alternating sweeps until a red object is seen in the clamp’s “field of vision” - contained within the overall work envelope of the arm
2: retrieveAndPlaceObject( ): which drops the arm a certain angle, clamps the object, picks up the arm a certain angle, rotates back to angle 0 (base position), drops the arm again, unclamps the object, and returns arm up to base position.

These two functions are called inside main using a global boolean “objectSeen” which indicates if the clamp’s colour sensor has seen a red object or not.

Coding up all of these constituted the entire range of motion needed for the project. All that needs to be done is test it on the robot, which Adam has.

Total hours worked: 5 hrs (Mynah)

DAILY LOG 2:
Mynah, Tue 21/3

Adam managed to test the code and send a video response of the robot arm doing very tiny sweeps, and not responding to setMotorTarget. I then coded up a test of two functions:

1: moveByTarget() which uses a basic setMotorTarget command to sweep right once and back to base 0 position
2: moveByTime() which uses a setMotorSpeed command to move the motor a positive speed for a set amount of time (so the arm swings right) and then a negative speed for the same amount of time (so the arm swings back)

The robot responded to the setMotorSpeed command, so I will have to migrate my code to this method later. The failure of this method is to specify angles succinctly - it just specifies a speed and time for the arm to move. We will have to figure out a way to record the angles. I had previously suggested using two gyroscopes at each joint, but we unfortunately did not have a spare gyroscope picked up to work with for the build at the time.

For the time being, my plan of action is to figure out a way to record angles using just the speed and time, and to migrate my code to use setMotorSpeed. Perhaps I will be able to use getMotorEncoder to record the angle turned by the motor in terms of its rotation. I am hoping that my other teammates will also respond and pitch in, as I have not heard from any of them yet. Further testing is needed to determine how to move ahead - so I will email the Professor requesting an extension as I have been mostly working on my own and the deadline is approaching very soon. I have also communicated my request for the other team members to chip in, and hope that they will respond soon.

Luke, Tue 21/3
Everyone but Mynah had a call discussing the necessary equations we'd have to perform for the robotic arm. We went over the slides provided to us reading through them to get a better understanding of what's required from us. We focused on going through the slides that involved Kinematics. Mostly forward kinematics which is the process of computing the position and orientation of the claw from the joint angles in the robotic arm. We also went through some example videos and did more research on other site of how to perform said equations.

Total hours worked: 2 hr

DAILY LOG 3:
Luke, Thur 23/3

Myself, Adam and Jon met up in person on campus to work on the robot. We modified the robot by moving the light sensor to the claw so we can detect the object and adding another gyro Sensor to check the angle of the arm. We had to completely rework the base code today and we started this by programming the robot to raise and lower the arm. Once we got that working we added a function to open and close the claw. With this combined we were able to pick up an object that was placed in a specific position. Due to the weight added to the end of the arm from the light sensor the arm's start position is down. This means the arm starts the program by powering the armMotor to raise upwards. Looking ahead to avoid any problems with the light sensors we close the claw so in future functions it doesn't interfere with when we're searching for the object.

Total hours worked: 3 hr

DAILY LOG 3:
Luke, Fri 24/3

Everyone but Mynah met up on campus. With the arm now able to pick up the object we started working on a way for the robot to detect the object. We programmed the robot to raise up, close the clamp and reset to the right by turning the arm till it reached the touchSensor in the bottom of the robot. With our start position in place we then worked on programming the robot to detect the object. With the arm positioned fully to the right it turns left till it finds the object or reaches -180 degrees. This took the majority of the time as if the arm was raised too high it can't detect it and if its too low it ends up knocking the object over. Once we were able to get it working we ended up with problems with the arm no longer being able to properly lift the object. While we were still trying to fix this I created a function at the end that places the object to the very right using commands from past functions. After much trial and error we managed to get it working and successfully recorded the video.

Total hours worked: 5 hrs 30 mins

DAILY LOG 4:
Mynah, Sat 25/3

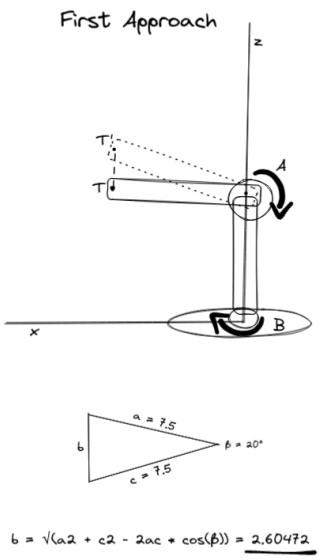
After everyone else met up and had the code tested and running, I went back over the code and modularised it to reduce code repetition. We are currently using a combination of setMotorTargets and gyro sensors for movement. Jon has been working on the rotational matrices extensively, and has written an additional Python script to model the forward kinematics movement as rotational matrices, along with a GeoGebra simulation. I believe we are nearly finished up, and I am glad that the other team members were able to meet and complete up the project at the end.

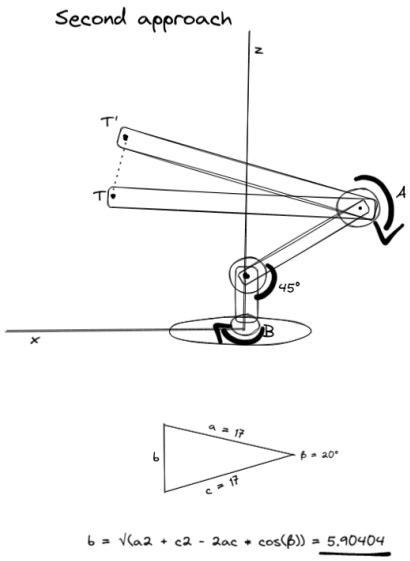
Total hours worked: 1hr

DAILY LOG 5:
Jon, Thur 23/3 - Sat 25/3

Utilization of rotational matrices:

Utilizing rotational matrices to calculate the position of the claw based on the feedback provided by the gyros proved to be quite a challenge at first.
Especially the bent part connecting the base of the robot and the forearm needed several attempts to get right.
First, we used a simplified approach to the structure of the robot, reducing the bend part to a straight link meeting the forearm at a 90 degree angle.
The rough schema can be seen in the image below titled “First Approach”.

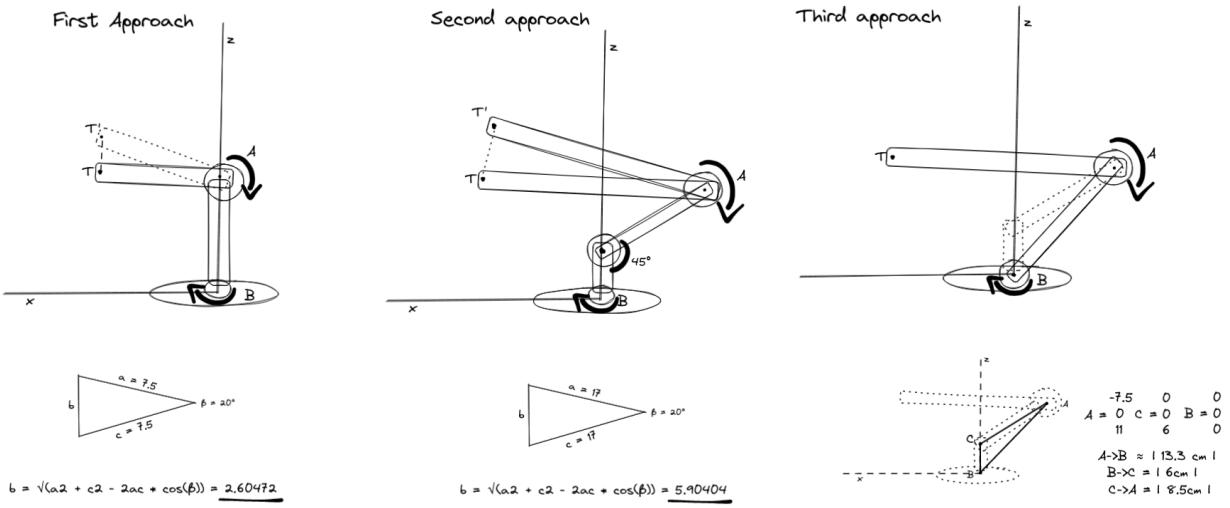




We then started to use rotational matrices to calculate the position of a claw given a certain rotation of the base motor.
Working with the orientation of the base was quite simple, because at this point we didn't consider the tilt of the forearm.
But once we started calculating the tilt of the forearm, the calculations showed to be off.
We recognize that this was due to the fact that our simplified model resulted in a shorter length of our robotic forearm. Which has quite an effect on the Distance travelled by the end effector as can be seen by the calculations above.

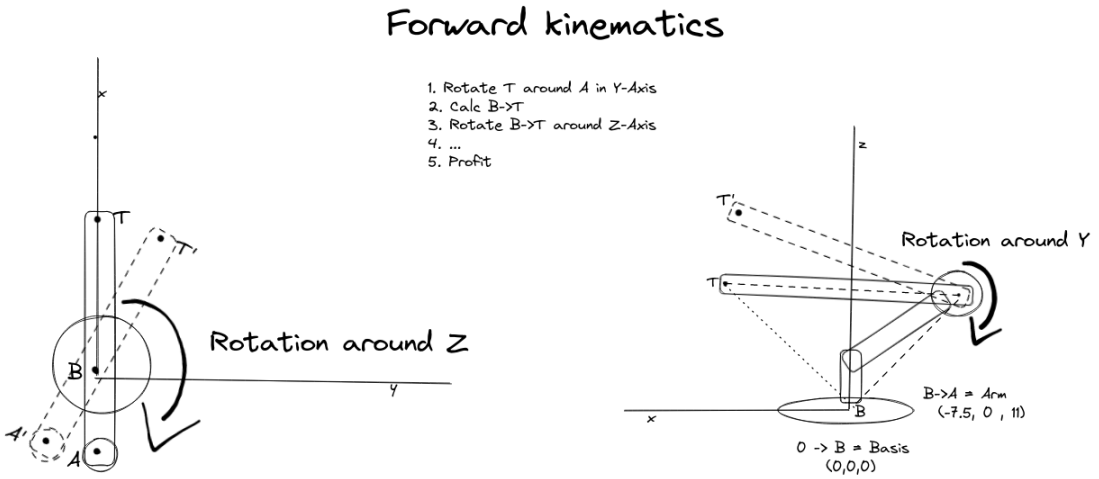
We then decided to include the bend part in our calculations, but weren't so sure how to actually do so.
All of our reading material and examples only included straight links.
Therefore, to get around that issue we decided to regard the bend in the link as a distinct motor separating two straight links, with a fixed rotation.
This proved to work well, but seemed to be an inelegant solution, so we decided to try and find a better way to do so…

We then had the idea not to regard the part as a bend separating to straight parts, but to simply regard it as a single straight part by flattening it into a single vector.

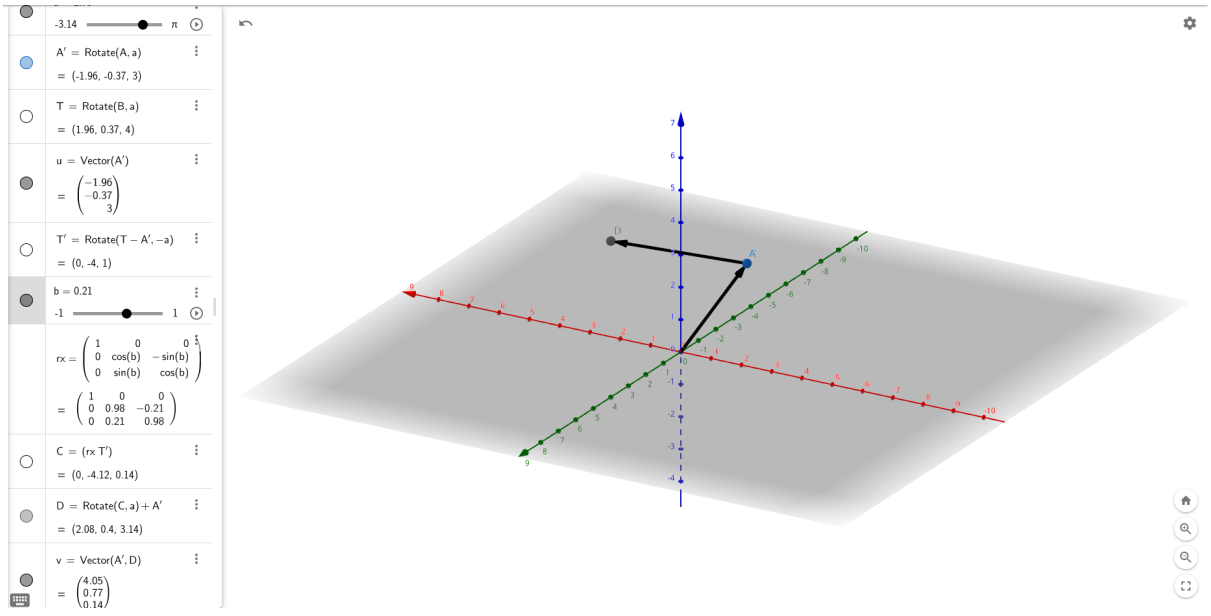


This proved to work well, but we still had to find a way to calculate the rotation of the forearm in combination with the rotation of the base.
Because we had started with calculating the orientation of the base, we first calculated this and then used the rotation of the forearm to calculate the final position of the claw.
But due to the Z rotation changing the alignment of the forearm in the X-Y plane, it was hard to calculate get the correct axis to rotate around.
First we tried to rotate the Z axis and projecting the arm back to the X axis to then calculate the rotation around the Y Axis, when we noticed, that simply

Rotating the Y axis first and then the Z axis would give us the same result. As the Rotation of the Y axis is not affected by the Z rotation in our case.

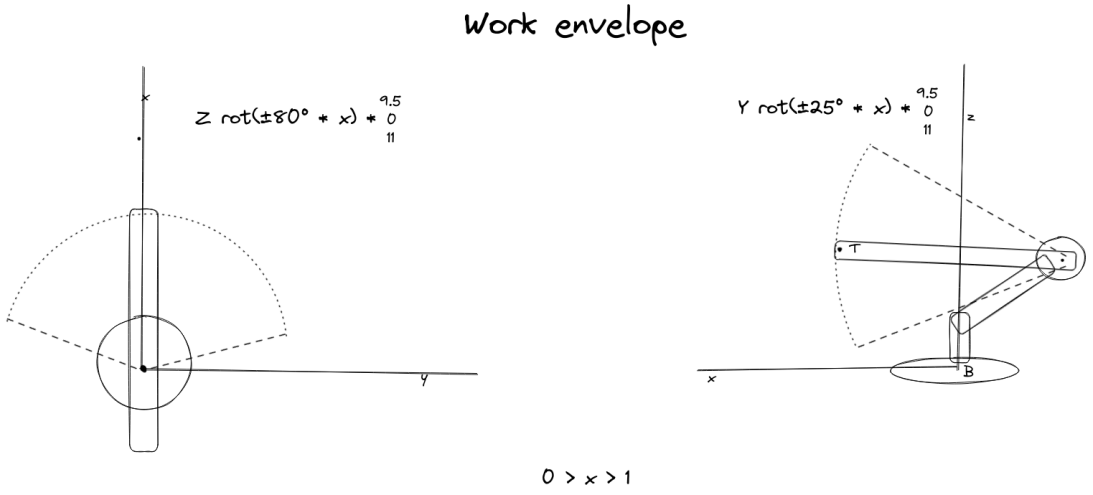


To better better visualize the process a GeoGebra simulation was created to show the process of calculating the position of the claw and its resulting movements.



At first the general code for the calculations was written in python, due to the fact that it has a better toolkit for matrix calculations than C.
But after the algorithm was working, it was rewritten in c to make it usable in the robot code.

I also created a visualization of the effective work envelope of the robot, which can be seen in the image below.



Total hours worked: 6 - 8 hrs over the last 3 days
Mixed with research and chipping in on other parts of project.