# Pokèmon Gotta' Catch 'Em All: Armlab

Yu-Tung Lin, *Masters Candidate* Theodore Nowak, *Masters Candidate* Zhentao Xu, *Masters Candidate*

***Abstract*—The goal of this project is to create a robotic gripper to "catch" Pokèmon. To accomplish this task a state machine was designed, a gripper system was constructed, blob detection and affine homography was implemented, and inverse and forward kinematics were computed. We evaluated each subsystem of our project on tasks emphasizing accuracy and repeatability. Methodology of every components were explained in detail. Tests for each components and the entire system were implemented and result were analysed.**

## I. Introduction

IN this project we sought to identify, grasp, and remove as many Pokèmon as possible in as short a time as possible. As such, a **blob detector** was implemented to detect Pokèmon on a level, 2 dimensional plane under a ceiling mounted camera. The blob detection task aimed to accurately distinguish Pokèmon from other objects in the scene via colour thresholding and contour size matching. As part of translating the detected pixels into the coordinate frame of the robotic arm, an **affine homography** was calculated to map the pixel frame to the world frame (and vice versa). Guiding the detection task, a **state machine** was designed to navigate the arm through the components of movement through waypoints, gripping, delivery, and seeking another target. To move from waypoint to waypoint inverse kinematics were applied when seeking to reach a desired location,and forward kinematics were implemented when checking for arrival. At the end of each trajectory the state machine would instruct the gripper either grasp or release.

Testing was done via experiments constructed to test individual subsystems. Detection methods were tested on the grounds of detection robustness, repeatability, and accuracy. Forward and inverse kinematics were tested to be reliable and accurate. Teaching and Repeat testing result were showed in a 3D plot where the whole trajectory along with way points were plotted. Systemic catch experiment are implemented and had a very good performance.

## II. Methodology

### A. State Machine

State machine was utilized as a method to systematically organize the motion and behavior of the Rexarm system. Also as is shown in Figure 1, the state machine consists of eight states, where two of them (in green color) was further decomposed into three sub-states. The switching between different states were implemented by a State Manager Class in Python.

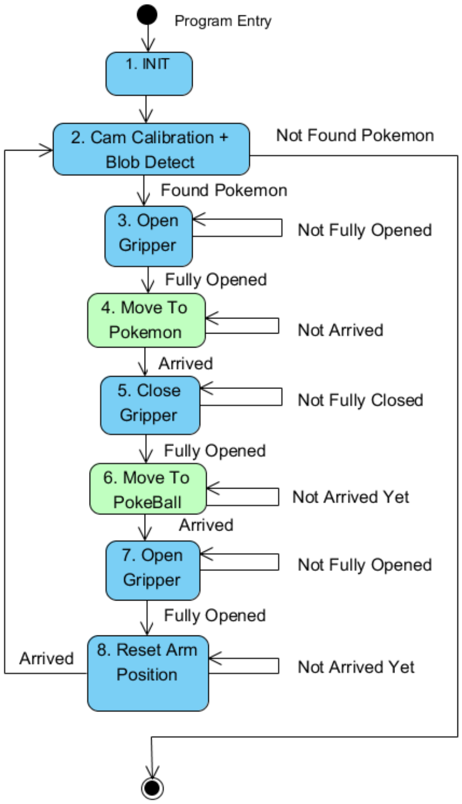


Fig. 1: State Machine Diagram

After executing the program, the first state program entered was "Camera Calibration", where program would wait for user to do camera calibration (See Affine Transform Section for detail). When camera calibration was done, program would automatically enter "Blob Detection" state, where the location of the next Pokèmon would be calculated (See Blob Detection Section For detail). The program would then enter the "Open Gripper" state, where the command of opening gripper would be kept sending to Rexarm until gripper was fully opened based on the angle feedback (See Gripper Design Section for detail). In the "Move To Pokèmon" state, Rexarm would move the gripper to Pokèmon through some way points. These way points would be calculated based

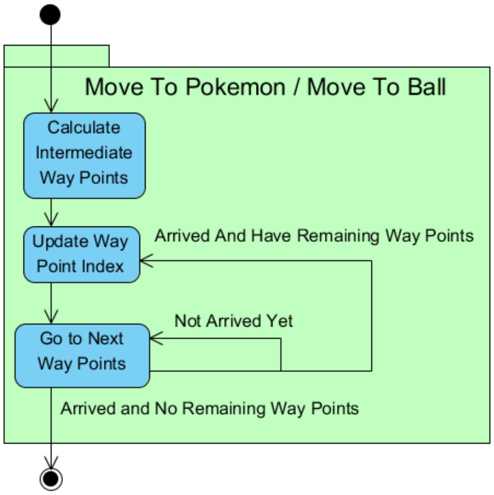

Move To Pokemon / Move To Ball
Calculate Intermediate Way Points
Update Way Point Index
Arrived And Have Remaining Way Points
Go to Next Way Points
Not Arrived Yet
Arrived and No Remaining Way Points

Fig. 2: Sub States of ”Move To Pokèmon / Pokeball”

on the location of Pokèmon and the initial position of gripper.(see Path Planing Section for detail) During this state, real-time location of gripper would be checked form time to time using forward kinematics and the feedback of joint angles from servo motors (See Forward Kinematics Section for detail). Once the location of Pokémon was achieved, program would then catch the pokémon during ”Close Gripper” state, move the caught pokémon to the location of Pokéball during ”Move To Pokeball” state, drop the Pokémon in ”Open Gripper” section, and eventually reset the Rexarm gesture during ”Reset Arm Position” state.

### B. Gripper Design

A gripper was used to catch five different Pokèmons. Each Pokèmon had different shapes and sizes. They were placed at a random distance from the Rexarm and they may be standing or laying down. Our goal was to design the gripper to be able to catch them from multiple orientations and positions and to simplify the mechanism was also a target to make the gripper move efficiently and stable.

Fig. 3: Venus Flytrap

Our design was inspired from a plant called ”Venus flytrap” (shown in Fig. 3 [5]). Venus flytrap could easily trap its prey not only by closing its leaf rapidly but also it had a teeth-looking feature around its leaf. Therefore, our appearance of the gripper was designed to be similar to Venus flytrap. Next, the size of the gripper was decided by measuring the dimensions of the Pokèmon. The length of the gripper was slightly longer than the largest Pokèmon. The width was a little wider than the radius of the Pokèmon. Finally, the height of the gripper was set to be the same height of the motor. The length of the teeth was chose to be long enough to reach the other side of the gripper when it was opened at a small degree so that the Pokèmon would not dropped out.

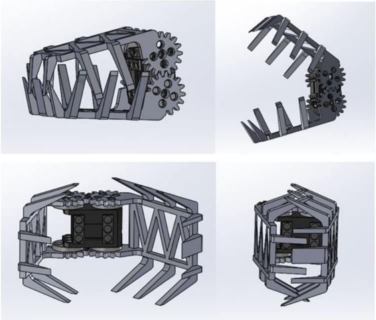


Fig. 4: Gripper CAD Design.

After the dimension of the gripper was determined, we tested the performance by actually catching the Pokèmon when the gripper was attached to the Rexarm. Then, we modified the gripper from the testing result. The details of the testing result were shown in next section. The latest version of gripper was shown in Fig. 4 and Fig. 5

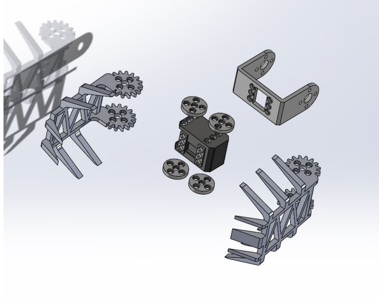


Fig. 5: BOM

The gripper needed to be attached to the Rexarm to finish the task. Fig. 6 showed the design of the connector. The wire could get through the rectangle space to connect to both motors.

To program the gripper, we first found the angle for the gripper to be opened and closed. We checked the feedback angle to see if it had already finished operating. If it still moving, the state machine would stuck until the

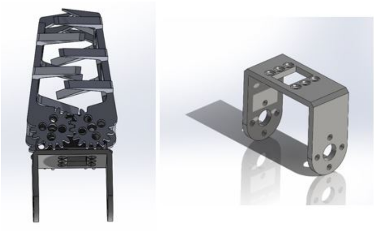


Fig. 6: connection between gripper and AX12 motor

feedback reached an acceptable error tolerance. Note that closing the gripper was complicated because a Pokèmon might kept the gripper opened then the state machine would never move on. To solve this problem, we added a timer. The timer would start when the command was set to close gripper, then after a certain amount of time (0.1sec), the gripper would close again and move on to the next state.

### *C. Affine Transform*

Affine homography was used as a special case of general homography to determine the mappings from the pixel frame to the world frame. Affine homography differs from general homography in that it only seeks to determine the translational and rotational mappings between coordinate frames as opposed to the aforementioned in addition to a perspective mapping sought by general homography. Perspective mapping is unnecessary in our case, as the z distance from the lens to the plane on which Pokèmon exist is constant.

To calculate the affine homography, the general homography in 3 dimensions is first reduced to two and further reduced by eliminating the projective transform as shown in Figure 7 [2].

$$\begin{bmatrix}x_w\\y_w\\z_w\end{bmatrix} = \begin{bmatrix} f_x & 0 & C_x & 0\\ 0 & f_y & C_y & 0 \\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}\begin{bmatrix} & R & \end{bmatrix} & \begin{bmatrix}T\end{bmatrix}\\ \begin{bmatrix} & P & \end{bmatrix} & 1\end{bmatrix}\begin{bmatrix}x_p\\y_p\\z_p\\1\end{bmatrix} \tag{1}$$

Fig. 7: General 3D Homography Matrix

Where $x_p$, $y_p$, $z_p$ is the 3D point in pixel coordinates. $f_x$, $f_y$, and $C_x$, $C_y$ are intrinsic properties of the camera. $R$, $P$, $T$ are the rotation, projection, and translation matrices respectively. And $x_w$, $y_w$, $z_w$, 1 are the homographic coordinates in world space. In Figure 8 the reduced 2D affine homography is shown. It can trivially be found by dropping out the z coordinates, setting the projection matrix to 0, and multplying out the two scalar matrices of coefficients.

$$\begin{bmatrix}x_w\\y_w\\1\end{bmatrix} = \begin{bmatrix} a & b & c\\ d & e & f\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x_p\\y_p\\1\end{bmatrix} \tag{2}$$

Fig. 8: Reduced 2D Affine Homography Matrix

Where $a$ through $f$ are scalar mapping coefficients derived from the multiplication of the camera properties and rotation-translation matrices. $x_p$, $y_p$ are the pixel coordinates in homographic coordinates. And $x_w$, $y_w$ are the world homographic coordinates. The final linear regression matrix is then given by rearranging the 2D Affine Homography matrix into the following construction. Which we can trivially solve via the Moore-Penrose pseudoinverse to establish the world-to-pixel-mapping homography coefficients given $n$ pairs of pixel and world coordinates.

$$\begin{bmatrix}x_{w_1}\\y_{w_1}\\\vdots\\x_{w_n}\\y_{w_n}\end{bmatrix} = \begin{bmatrix} x_{p_1} & y_{p_1} & 1 & 0 & 0 & 0\\ 0 & 0 & 0 & x_{p_1} & y_{p_1} & 1\\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots\\ x_{p_n} & y_{p_n} & 1 & 0 & 0 & 0\\ 0 & 0 & 0 & x_{p_n} & y_{p_n} & 1\end{bmatrix}\begin{bmatrix}a\\b\\c\\d\\e\\f\end{bmatrix} \tag{3}$$

Fig. 9: Reduced 2D Affine Homography Matrix

Thus upon initiation, to gain a mapping of pixel to world coordinates for both the blob detection task and inverse kinematics tasks to be described later, we first solve the mapping given in Figure 9 using $n = 6$ pairs of points to retrieve the mapping coefficients. This proved favourable when compared to the calibration matrix derived by OpenCV via the **getAffineTransform()** function as the OpenCV implementation was hard-coded to received 3 pairs of pixel and world coordinates, and thus exactly matched the provided data [1]. Figure 9 allows for optimal fitting to equivalent data given more points, and thus averages the location to achieve a better fit.

### *D. Blob Detection*

After calculating the linear mapping matrix via the affine homography given above, blob detection was applied to detect the various Pokèmon by their colours. This task was implemented using various functions given by the OpenCV computer vision library [3] [4]. The library was used to edit the live video stream of the camera above the Rexarm in a number of ways.

First, all black masks were applied to areas of a live video frame in which Pokèmon should not be

**Algorithm 1** Blob Detection Process Chain

```
Detect ($P_c$ | $T_c$, $M_a$, $M_b$, $M_p$, $M_e$)
Where $P_c$ - Pokèmon of Colour c, $T_c$ - Threshold for Colour c, $M_a$ - Arm Mask, $M_b$ - Base Mask, $M_p$ - Pokèball Mask, $M_e$ - Edge Mask

for $P_c = 0$ to 5 do
    Read one video frame
    Convert to HSV
    applyMask($M_e$)
    applyMask($M_a$)
    applyMask($M_b$)
    applyMask($M_p$)
    applyBinaryThresholding($T_c$)
    applyOpenFiltering(Small Kernel)
    applyClosedFiltering(Medium Kernel)
    applyOpenFiltering(Large Kernel)
    applyClosedFiltering(Large Kernel)
    findContours()
    for # of contours do
        Find minimum enclosing circle
        if Radius of circle is Pokèmon sized then
            Store properties
            Recursively find shortest distance to base
        end if
    end for
end for
return Center point of Pokèmon closest to base
```

detected, such as outside the table area, where the arm exists, where the base of the arm exists, and where the Pokèball exists. This prevented the system from detecting the arm itself (which was similar in colour to the dark blue Pokèmon) and from re-detecting deposited or external Pokèmon. Next, the binary colour threshold was applied, followed by a series of filters. The two filter's used were an Open Filter, and a Closed Filter [3]. The Open filter removes stray points via an Erosion followed by a Dilation, while the Closed Filter did the opposite, solidifying otherwise patchy positive binary areas. The kernel sizing chosen optimizes the filtering of noise while resulting in large solid shapes where Pokèmon exist, all the while filtering out the blue in the Rexarm. One can imagine an iteratively larger kernel first removing small, granular pixel areas, then returning to solidify into larger areas, that which remains. To complete the process, contours were drawn around such areas, minimum enclosing circles were constructed, and those of the right size were identified as Pokèmon.

### *E. Inverse Kinematics*

Inverse kinematics is needed to navigate the Rexarm to the location of Pokémon. Due to the simple mechanical strcture of Rexarm, graphical method was used to derive the joint angles coorsponding to a certain end effector's location $(x, y)$ and tilt angle $\phi$. Frame $\{Ground\}$, Frame {Base} and symbols are illustrated as in Fig. 11 and in TABLE I. The solution of inverse kinematics were not unique and at most 4 solutions could be got. In this lab we simply picked the solution with $\theta_{shoulder} >= 0$ and "elbow up" configuration, because it could be easily proved that this solution could reach the maximum range on the board and cause no ambiguity in derivation.

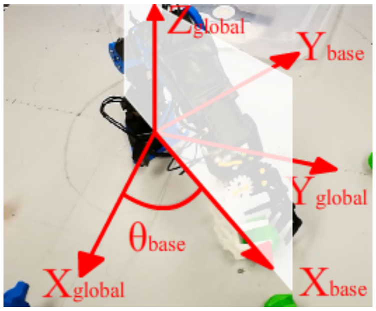


Fig. 10: Symbols for Calculation of Inverse Kinematics

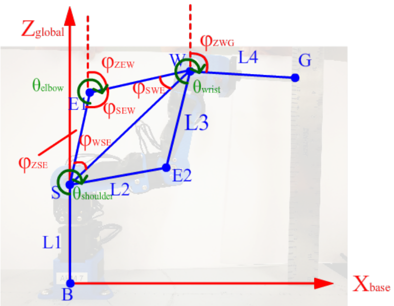


Fig. 11: Symbols for Calculation of Inverse Kinematics

TABLE I: Symbol Table

| Symbol | Unit | Description |
|---|---|---|
| $L_1$ | [mm] | length of the first link from ground to shoulder |
| $L_2$ | [mm] | length of the second link from shoulder to elbow |
| $L_3$ | [mm] | length of the third link from elbow to wrist |
| $L_4$ | [mm] | length of the forth link from wrist to end effector |
| $\phi_{ZSE}$ | (rad) | angle between Z-axis and $L_2$ Link |
| $\phi_{ZEW}$ | (rad) | angle between Z-axis and $L_3$ Link |
| $\phi_{ZWG}$ | (rad) | angle between Z-axis and $L_4$ Link (tilt angle) |
| $\phi_{SEW}$ | (rad) | inner angle between $L_2$ and $L_3$ |
| $\phi_{SWE}$ | (rad) | inner angle between $L_{SW}$ and $L_3$ |
| $\phi_{WSE}$ | (rad) | inner angle between $L_{SW}$ and $L_2$ |
| $\theta_{base}$ | (rad) | base servo motor joint angle |
| $\theta_{shoulder}$ | (rad) | shoulder servo motor joint angle |
| $\theta_{elbow}$ | (rad) | elbow servo motor joint angle |
| $\theta_{wrist}$ | (rad) | wrist servo motor joint angle |

The joint angle of base servo motor dependeded only on $x, y$ components of target by the Eqn. 4. Note that the solution $2\pi$ at singularity was removed from solution to prevent ambiguity. The locaiton of wrist can be uniquely derived using Eqn. 5. Distance between shoulder joint and wrist joint is calculated using Eqn. 6. Since for triangle $SEW$, the length of all its three edge are known ($L_2, L_3, L_{SW}$), all its inner angles could be calculated using Eqn. 7. The angle $\phi_{ZSW}$between $Z$ axis and $SW$ is calculated using Eqn. 8. Note since we only picked "elbow up" configuration, so only joints $E_1$ is needed for calculation. The final result can then be derived in Equation 9

$$\theta_{Base} = Atan2(G_y, G_x) \in (-\pi, \pi) \tag{4}$$

$$\begin{bmatrix} W_x \\ W_z \end{bmatrix} = \begin{bmatrix} G_x - L_4 \cdot \sin(\phi_{ZWG}) \\ G_z - L_4 \cdot \cos(\phi_{ZWG}) \end{bmatrix} \tag{5}$$

$$L_{SW} = \sqrt{X_W^2 + (Z_W - L_1)^2} \tag{6}$$

$$\begin{cases} \phi_{WSE} = \frac{L_{SW}^2 + L_2^2 - L_3^2}{2L_2 L_{SW}} \\ \phi_{WES} = \frac{L_2^2 + L_3^3 - L_{SW}^2}{2L_2 L_3} \\ \phi_{WSE} = \pi - \phi_{WES} - \phi_{WSE} \end{cases} \tag{7}$$

$$\phi_{ZSW} = Atan2(W_x, W_z - L_1) \tag{8}$$

$$\begin{cases} \theta_{Shoulder} = \phi_{ZSW} - \phi_{WSE} \\ \theta_{Elbow} = \pi - \phi_{SEW} \\ \theta_{Wrist} = \phi_{ZWG} - \phi_{ZEW} \end{cases} \tag{9}$$

### F. Forward Kinematics

Forward kinematic was used to determine the location and orientation of end effector given joint angles. We strictly followed Devanit-Hartenberg Convention in setting up the coordinates as is shown in Figure 12. Based on the frame, four parameters for each joints were calculated and listed in TABLE II. Using the Change of Frame formula, the expression of location and tilt angle of end effector with respect to ground frame is expressed in Eqn.

TABLE II: Denavit-Hartenberg Table

| Link | $a$[mm] | $\alpha_i[(rad)]$ | $d_i$[mm] | $\theta_i[(rad)]$ |
|---|---|---|---|---|
| 1 | 0 | 0 | 116 | $\theta_{base}$ |
| 2 | 0 | $-\frac{\pi}{2}$ | 0 | $\frac{3\pi}{2} - \theta_{shoulder}$ |
| 3 | 100 | 0 | 0 | $-\theta_{elbow}$ |
| 4 | 100 | 0 | 0 | $-\theta_{wrist}$ |

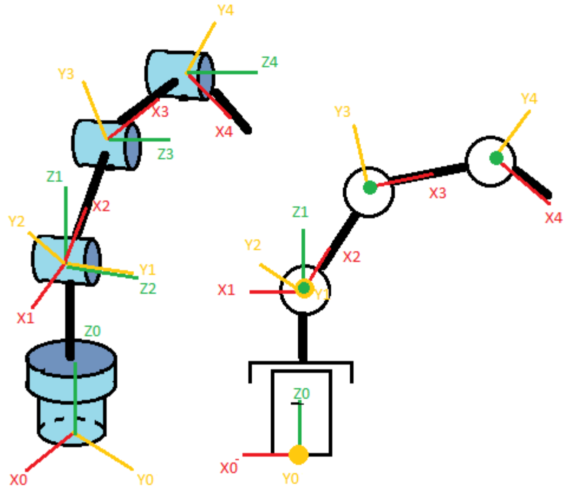


Fig. 12: Forward Kinematics Coordinate Frame Assignment

$$\begin{cases} P^0 = R_1^0 R_2^1 R_3^2 R_4^3 P^4 \\ \quad = \begin{bmatrix} C_b & -S_b & 0 & 0 \\ S_b & C_b & 0 & 0 \\ 0 & 0 & 1 & 116 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -S_s & C_s & 0 & 0 \\ 0 & 0 & 1 & 0 \\ C_s & S_s & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ \quad \begin{bmatrix} C_e & S_e & 0 & 100 \\ -S_e & C_e & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C_w & S_w & 0 & 100 \\ -S_w & C_w & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 100 \\ 0 \\ 0 \\ 1 \end{bmatrix} \\ \phi_T = \theta_{shoulder} + \theta_{elbow} + \theta_{wrist} \end{cases} \tag{10}$$

### G. Path Planning

The main idea of path planning was to control the path Rexarm went through, as well as gripper's gesture when gripping Pokémons. The quickest way was inserting intermediate way points in space and let Rexarm go through them one after the other. However, three essential problems need to solved: how to prevent collision, short to minimize the time, and how to reach the maximum space. The solution to both questions are stated below.

*1) Prevent Collision:* Generally speaking, in order to prevent collision, locations of obstacles in space should be know by the program. However, in this lab, all Pokḿons were on the board, which is a 2D plane. Therefore, as long as Rexarm's motion space was higher then the board by at least the height of Pokèmons, no collision would occur. Therefore we inserted two way points, one right above Pokémon and the other right above Pokeball.

*2) Choose Shortest Path:* Time was taking into account in the path planning. Since the Pokéball is at the singularity point of base servo, choosing proper dropping location will greatly short the time. The strategy we used was that, we set the dropping location based on where Pokémon was grabbed. When it's grabbed on the left half circle, the dropping location would also be set on to the left of singularity.

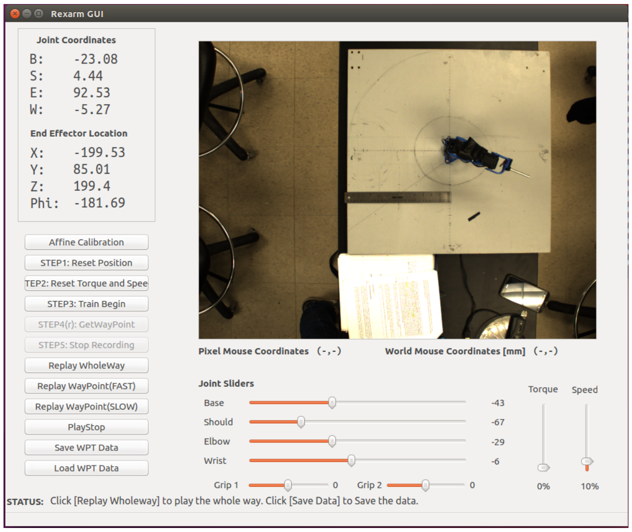
*3) Tilt Angle Plan:* We chose the tilt angle so that the reachable space (workspace) was maximized. One strategy we utilized was that we calculated the tilt angle based on the distance from gripper to the base. When the distance was large, gripper tended to be flat $\phi \approx 0$ , then distance got closer, gripper tended to be vertical $\phi \approx \frac{\pi}{2}$; when the gripper was very close to base, it tended to incline to base $\phi > \frac{\pi}{2}$. Some critical distance were chosen based one test, and linear function was fitted to for calculating the tilt angle.

### H. Teach And Repeat

The main idea of teaching Rexarm is to record the sensor feedback of every motors of every way points and use these feedback values as commands when replaying.

To teach the Rexarm play with the board game, we had to set the motor torque to be zero first. Then, we recorded both trajectory of the whole process and the way points lower and above of each holes by clicking a button on GUI (Figure 13 ) for each point. After the button was triggered, the program would save these value to a list. After teaching it once, we save all joint anles in the list to a file. Load these files when we wanted the Rexarm to replay. If we wanted it to operate in high speed mode, then set the speed command to a higher percentage. Otherwise, set the speed low. Also, remember to set the torque back before replaying. When the Rexarm was reaching its goal, we had to check whether it was arrived or not. Since there must exist an error between the sensor values and our command, we set an acceptable error tolerance to each joints. If Rexarn was within this tolerance, then it could start to move toward the next way point. How to pick the way points and what the result was would be described in next section.

Although we didn't included the teaching and repeating code in the final tar file, Figure 13 shows the control panel of teaching and repeating. Buttons were enabled based on the current states for the purpose of user friendly, slider bar would change values based on feedback joint angles to ensure safety. Saving and loading data function were useful and easy to operate.



Joint Coordinates

B: -23.08
S: 4.44
E: 92.53
W: -5.27

End Effector Location

X: -199.53
Y: 85.01
Z: 199.4
Phi: -181.69

Pixel Mouse Coordinates (-,-)

World Mouse Coordinates [mm] (-,-)

Base -43
Should -67
Elbow -29
Wrist -6

Grip 1 0
Grip 2 0

Torque 0%
Speed 10%

STATUS: Click [Replay Wholeway] to play the whole way. Click [Save Data] to Save the data.

Fig. 13: The GUI for Teaching and Repeating.

## III. Experimental Result

### A. Gripper

Four versions of gripper would be introduced one by one. Strength would be remained in the next version while the weakness be removed. The performance of the gripper would be tested with other components in the later section of System Testing.

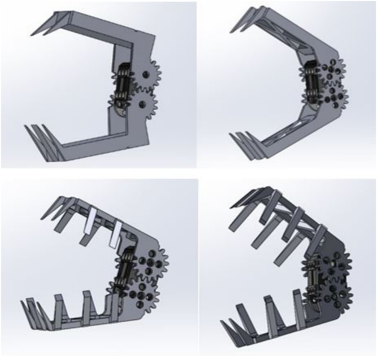


Fig. 14: Four versions of opened gripper.

The first version of the gripper had teeth on the outside edge shown in the upper left in Fig. 14. It could successfully grasp the Pokèmon only when foams were utilized on the inner side of gripper. However, this design was too heavy and needed external materials which would increase the load of the motor when operating. Moreover, the length of the gripper was too long, and the space bounded by the closed gripper was too large that Pokèmon could fall out. Furthermore, this gripper could not connect to XL320 motor because there were no holes for the rivets.

After reducing the length, shifting both sides to be closer and adding holes for rivets, we had gripper version 2 shown in the upper right of Fig. 14. Both sides of the gripper were also partly cut out as a window to make it lighter. However, foams were still needed to hold Pokèmon in the gripper. There were redundant structure that weights a lot, so we then made the width for each side decreased to be 3mm, which was strong enough and weight less. The window size on both sides increased, and this lowered the probability of dropping Pokèmon because the space was more flexible for Pokèmon to get inside. The major change for this version was that the teeth were added on every edges. This feature made our gripper grasp Pokèmon easily and held them tight in the gripper. Although the teeth worked successfully, they were too fragile. Gripper version 3 was shown in the lower left of Fig. 14.

Eventually, the teeth were made thicker and larger, and the pattern of the window was redesigned as shown in the lower right of Fig. 14. It could easily grasp Pokèmon from different orientation no matter the Pokèmon was standing or laying down. The weight was light and the motor operated efficiently with our gripper.

### B. Calibration and Blob Detection

Six points were used to run the affine transform: (0,0), (0,280), (280,0), (0,-280), (-280,0), (150,150). As aforementioned this led to improved results over the OpenCV **getAffineTransform()** function which used three. The calibration achieved by the our affine transform was evaluated simultaneously with our blob detector. Pokèmon were placed 10cm apart in measured locations and blob detection consistency and accuracy was evaluated.

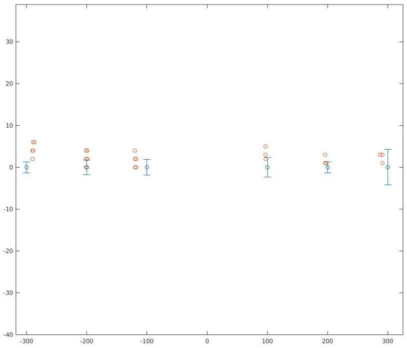


Fig. 15: Variance across Y axis during blob detection of green Pokèmon. Individual measurements in red, desired location blue, combined norm of x-y variance error bars shown.

As can be seen above, there was slight error and variance during blob detection. The mean l2 norm of the x-y variance was 3.0022 mm. This value is small enough to be considered insignificant due to the robustness of the gripper as shown. Variance in the blob detection was tested alongside accuracy of the affine homography. Error in the homography is indistinguishable when compared to inaccuracies from Pokeèmon placement and blob detection.

To support the blob detection task, and discover the adequate values in HSV colorspace needed to threshold the various coloured Pokèmon, a separate script was written which recorded the minimum and maximum HSV values of pixels clicked via the mouse. Using this script, and an accompanying visualization displaying detected contours from live video, colour thresholds were able to be accurately calculated. The table below indicates that out of 360 detections(5 Pokèmon * 3 distances * 3 detections * 8 positions) there were only 3 errors. All three of which occurred during testing with laying down Pokèmon, and all three of which were double counts of a single Pokèmon. Blob detection of standing Pokèmon was 100% accurate.

TABLE III: Blob Detection Accuracy

| | LB | DB | O | Y | G |
|---|---|---|---|---|---|
| Standing | 100% | 100% | 100% | 100% | 100% |
| Fallen | 100% | 100% | 98.6% | 100% | 97.2% |

### C. Inverse Kinematics

To see how accurate our Inverse Kinematics was, we tested with probe tip instead of the gripper. We set a plane 2cm above and parallel to the board. We clicked on the video on GUI to set our desired coordinate for the Rexarm to calculate the joint angles and move. After the Rexarm reached to the position, we measured the error between our desired coordinate and the probe tip by ruler. The heatmap of the result was shown in Fig. 16. The error became larger as the Rexarm stretched out.

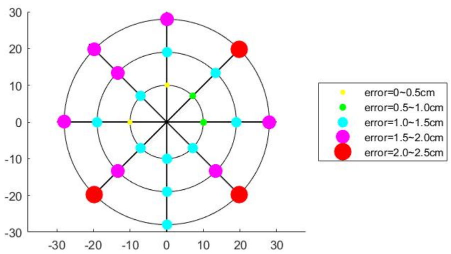
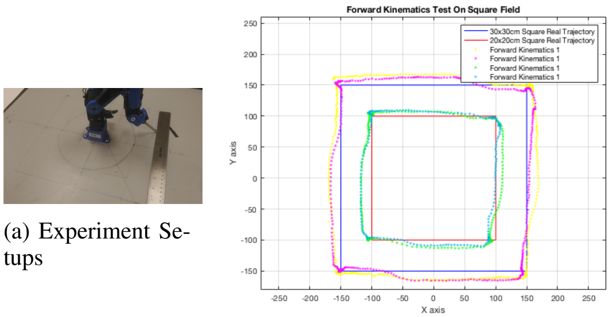


Fig. 16: Inverse Kinematics accuracy

### D. Forward Kinematics

Forward Kinematics was tested by comparing end effectors' real locations with the locations calculated using forward kinematics. In one test, we manually pull the tip of end effect along the edge of an 15cm by 15 cm and an 10cm by 10cm square trajectories. Rulers were used to ensure the straight real trajectory. (See Figure 17 (a)) The forward kinematics is then calculated and plotted on the same figure using MATLAB, as is shown in Figure 17 (b). There exist some errors in forward kinematics, but is relatively small and acceptable.



(a) Experiment Setups

(b) Forward Kinematics Result

Fig. 17: Forward Kinematics Accuracy Test Experiment Setups and Results

### E. System Testing

There were two systems to test, one was teach and repeat, and the other was catching Pokèmon.

*1) Teach And Repeat:* Way points were chosen to prevent bumping to the board. At first, we chose a way point into the board and an initial position before attempting to poke the hole. We then found these were not a good set of way points because the Rexarm move linearly from a point to another. Thus, it would pull out of the hole or trying to get to the position under the board. After a few tries, we realized that the optimal way of setting way points was to add points before and after entering a hole, so for each hole we have three way points. Moreover, the path of moving would be more accurate if we set the speed under 30%. Fig. 18 demonstrated how our Rexarm moved to desired way points in high and low speed. The result showed that the error in between was small.

*2) Catch Pokèmon:* Since Pokèmon had different poses on the board(i.e., standing straight, laying down, facing anywhere), we would discuss the result of catching Pokèmon in four different conditions. We tested ten runs for each circumstances.

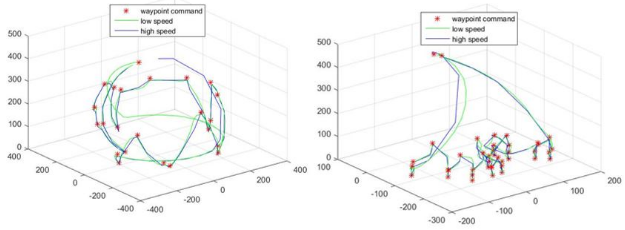


Fig. 18: Teach and repeat test

*a) Standing Pokèmon:* This condition was the easiest for our gripper to grasp Pokèmon. The testing positions for Pokèmons displayed as the Fig.19 shown. The result stated that our Rexarm could be almost 100% catching the Pokèmon at almost every positions. The failure were caused by Bulbasaur standing next to the arm because it was the tiniest Pokèmon that made the gripper hard to reach.

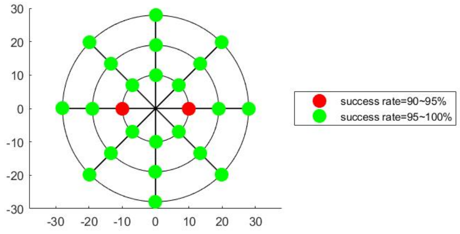


Fig. 19: Success catching rate: Pokèmon standing

*b) Laying Pokèmon Facing Towards Arm:* We found that it was hard for our gripper to catch Pikachu and Squirtle in this condition. For other Pokèmons, they were not problems. This was due to the size and shape of their head, theirs were relatively small and could easily slip out of our gripper at a far distance as shown in left of Fig. 20.

*c) Lying Pokèmon Facing Away from Arm:* The result (right of Fig. 20) of this condition seems randomly. Our gripper still worked fine in most cases.

*d) Lying Pokèmon parallel to the Arm:* Our gripper could also easily catch Pokèmon with this condition as in Fig. 21. For all the cases, the percentage of catching far Pokèmon is slightly lower than the near ones.

## IV. Conclusion And Future Work

In conclusion, we finished all the required tasks from this lab, including gripper designing, inverse kinematics

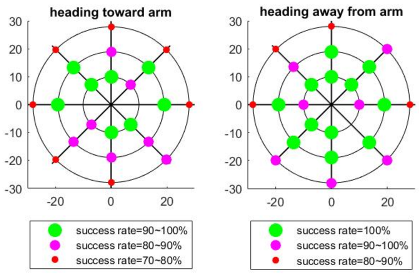


Fig. 20: Success catching rate: Pokèmon laying down

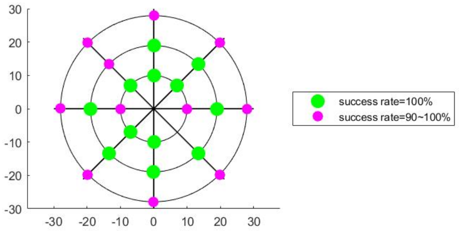


Fig. 21: Success catching rate: Pokèmon laying parallel to the arm

and forward kinematics calculation, teaching and repeating demonstration, catch Pokémon experiments, and testing of every parts and the entire system.

Future works includes improving the accuracy of the system, including changing the mechanical structure and dimension measurement for inverse and forward kinematics. Also, the time in the state changing is relatively long, potential reason might error Torrence still need to be further tuned.

## Appendix A
## Gripper Design

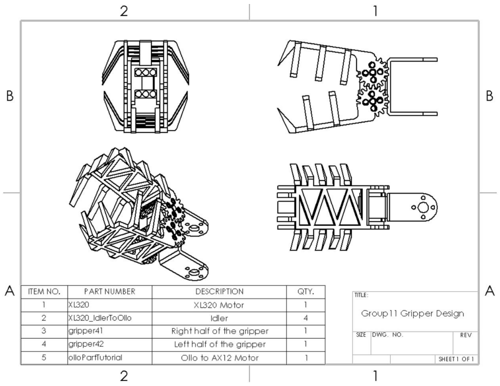
### A. BOM

Fig. 22 was the drawing sheet for the gripper.

1) 4 idlers
2) 1 pair of gripper
3) 1 otto
4) XL320 motor

### B. Assembly instructions

This section showed the steps of assembling the gripper.



| ITEM NO. | PART NUMBER | DESCRIPTION | QTY. |
|---|---|---|---|
| 1 | XL320 | XL320 Motor | 1 |
| 2 | XL320_IdlerToOllo | Idler | 4 |
| 3 | gripper41 | Right half of the gripper | 1 |
| 4 | gripper42 | Left half of the gripper | 1 |
| 5 | olloPartTutorial | Ollo to AX12 Motor | 1 |

Fig. 22: Gripper Drawing

1) use rivets to attach four circle idlers on the motor
2) put one side of the gripper on the idlers and match the holes to the idler
3) put the other side on and be aware of how two sides of the gripper aligned
4) put the ollo on the motor to the opposite side of the gripper

## References

[1] http://docs.opencv.org/2.4/modules/imgproc/doc/geometric$_transformations.html$
[2] https://april.eecs.umich.edu/courses/eecs467$_w14/wiki/images/4/44/Homographie$
[3] http://docs.opencv.org/3.0-beta/doc/py$_tutorials/py_imgproc/py_morphological_ops/p$
[4] http://docs.opencv.org/2.4/modules/core/doc/drawing$_functions.html$
[5] https://en.wikipedia.org/wiki/Venus$_flytrap$

**"I participated and contributed to team discussions on each problem, and I attest to the integrity of each solution. Our team met as a group on [Monday/Wednesday 09/09/16 - 10/14/16. Additionally, the entire weekend of 10/08/16 among various other dates]."**

**Theodore Nowak** **10/14/2016**

10/14/2016

Zhentao Xu 10/14/2016

Zhentao Xu 10/14/2016

Yu-Tung Lin 10/14/2016

10/14/2016