# Implementation of a 6-DOF Robotics Arm

Xiran Bai[1], Jiaxi Chen[1], and Yu Han[1]

***Abstract*— Robotic arms are widely used in harsh environment doing repetitive and complicated tasks in replace of human hands. As a result, increasing number of researches are dedicated to enhance the performance of robotic arm system. The goal of this project is to engineer and program a 6 degree of freedom (DOF) robotic arm system to distinguish color blocks and manipulate them to desired location autonomously. This is accomplished by integrating perception, acting and reasoning of the arm system. A blob detector is implemented to locate the block position. A gripper is designed to perform the gripping motion. Inverse kinematics is calculated to compute the joint angles. Then the state machine is applied to navigate the arm to designated locations. Our robotic arm demonstrated high level of accuracy and autonomy by successfully accomplishing 5 stacks and line up tasks. Further improvements include block orientation detection, and a more robust collision check could be made in the future.**

***Index Terms*— Robotics, Manipulator, Kinematics, Robotic arm**

## I. INTRODUCTION

Robotic arms are used extensively in industrial robotics and automations. The successfully programed robotic arm serves as an important tool for perceiving and manipulating objects at distance. This project provides an overview of the three main components in implementing an robotics arm, including perception, acting and reasoning. In this project, a 6 DOF robotics arm was built and programmed to manipulate wooden blocks with different colors. The robotic arm together with camera and depth sensor enable the system to perceive the color of the block, locate the position, grab, navigate through obstacles, and place it in the designated area.

In this robotic arm system, a gripper is designed as the end effector to perform gripping motion. The perception of color and block contour is achieved by a blob detector which distinguish the block from other objects by color thresholding and minimum area contouring. The positions of the blocks are determined by finding the moments of the block contour, and then relate the pixel position to world coordinates by several transformations. This is discussed in details in the Perception section. Once the block position is located, a state machine is designed to grab, navigate and deliver the block at the designated position. The inverse kinematics is applied during the process to make the arm move through waypoints, and the forward kinematics is implemented to return the current end effector position. The details of the methods are shown in Acting and Reasoning section.

The robotic arm system is tested on 5 different tasks, which are described in Reasoning and Results section. The successful completion of all tasks shows the robustness and accuracy of our system.

[1] Master of Science in Mechanical Engineering at University of Michigan, Ann Arbor

## II. MATERIALS AND METHODS

### A. Materials

The robotic arm system used in this project include a rexarm, kinect camera and depth sensor mounted directly above it. The rexarm has four joints and a tool tip as the end effector initially. The tool tip is then replaced by a 3D printed gripper designed for this project. The arm is equipped with 2 Dynamixel MX28, 2 AX12 servos for the joints, and 2 XL320 servos for the gripper. The Rexarm driver communicates with the arm using RS485 serial protocol through USB port and with Python program through Lightweight Communications and Marshalling (LCM) tools.

### B. Perception

To locate and distinguish the blocks precisely, two perception techniques are implemented, including camera calibration and blob detection.

1) *Camera calibration*

To relate pixel locations with real world coordinates, the camera calibration is performed. Because the system also incorporate depth sensor, there are some additional steps in calibration, affine transformation to align RGB with depth image, pixel to 3D camera frame transformation, and camera to work frame transformation.

The first one is to align pixel values in RGB image with depth image. In this case, an affine transformation is sufficient, because only the 2D pixel values are transformed. It is a special case of the general homography without the z coordinate. The 2D affine transformation matrix is described in equation (1), where $x_w, y_w$ are real world coordinates and $u, v$ are pixel values, element a through element f are matrix coefficients [1].

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{1}$$

The linear system is rewritten into the form of $Ax = b$, then performed pseudo inverse to recover the unknown parameters. The linear system is shown in equation(2). When number of points is more than 3, the method of least squares is performed using the built in function in Numpy [2]. In our case, we used 4 vertex points of the board in the RGB image and 4 points at the corresponding location in the depth image

to calculate the affine transformation.

$$\begin{bmatrix} u_1 & v_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & u_1 & v_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_1 & v_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & u_n & v_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \\ \vdots \\ x_n \\ y_n \end{bmatrix} \quad (2)$$

Compared to the calibration matrix got from OpenCV via getAffineTransform() function which only takes 3 pairs of points, our method provides a better measurement because it takes more points, returns the least square solution and thus gives a better fit [3]. Because the original depth frame is a little blur, sometimes it is hard to click the position at the right place. In order to help collect the points in the depth frame better, the pixel intensity in the depth frame is enhanced by 6 times. The result of the enhanced depth frame is shown in Figure 1.

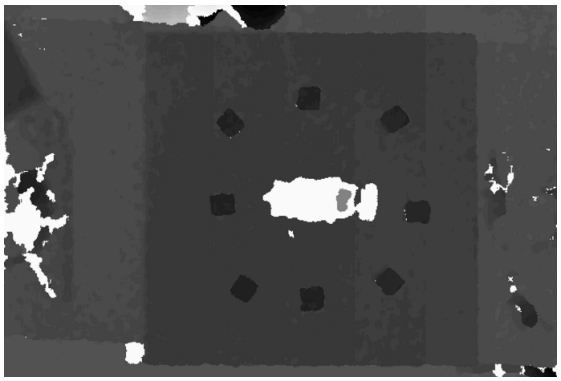


Fig. 1. To help with the collection of points using in the affine transformation, the contrast of the video in the depth frame is increased by changing the intensity of the pixel value.

After the RGB and the depth frame are aligned, the pixel values in either frame are then transformed into the camera frame using the camera intrinsic matrix, which is shown in equation(3). $F_x, f_y$ are focal length, and $c_x, c_y$ are principal point offset, which are all intrinsic properties of this specific camera.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = z_{(u,v)} \begin{bmatrix} 1/f_x & 0 & -c_x/f_x \\ 0 & -1/f_y & c_y/f_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \quad (3)$$

The $z_(u, v)$ term is described in equation(4). It relates the 2D pixel value with depth, and also converts the 10 bits depth d to meaningful units.

$$z = 0.1236 \tan(d/2842.5 + 1.1863) \quad (4)$$

The last step is to transform camera frame to workstation frame. Due to the camera is directly above the workstation, the axes of the two frames are aligned, and the values are at a known distance away. Therefore, another affine transform can be applied. In our case, 4 points in the camera frame with known workstation coordinates are collected to compute the affine matrix. After this step, the pixel values in the video are related to 3D real world coordinates with the origin at the base of the rexarm. 2) *Blob Detection*

To detect various blocks by their colors, blob detection is implemented using functions provided in OpenCV, which is a computer vision library [3]. The algorithm is summarized below.

**Algorithm 1** Blob detection algorithm

**Input:** color list, color threshold
**Output:** real world coordinate of the block center (centerWorld)

```
capture one video frame
convert the RGB image to BGR
convert BGR to HSV
for every color do
    apply color threshold
    apply erosion filter
    apply close filter
    findContours()
    for every contours do
        find the minimum enclosing rectangle
        find the moment of the rectangle
        find the size of the rectangle (sizeRec)
        if minSize < sizeRec < maxSize then
            compute centerWorld
            if armPosition < centerWorld < workStationBroader then
                store values
                drawContours()
            end if
        end if
    end for
end for
return centerWorld
```

First, the frame captured is converted from RGB to BGR then to HSV because OpenCV uses a format where pixels are stored in BGR. Next, a color filter returns a binary color mask using the HSV color threshold. To ensure the accuracy of the color threshold, a different function is implemented where the HSV values of the mouse pixel position are returned, which is shown in Figure 2. The erosion filter is

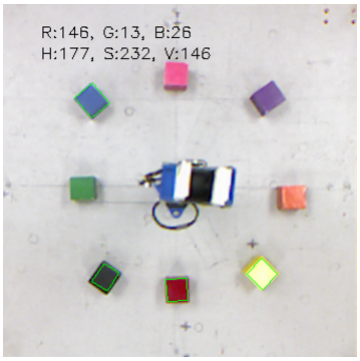



Fig. 2. The HSV value of the pixel location is shown on the top left of the picture as the mouse moves

then applied to shrink the foreground, followed by the close operation which is used to close holes in the foreground to further reduce noises. The kernel sizes are chosen to optimize the noise filtering and increases the robustness of the blob detector. Once the contours of color are found, the minimum enclosing area, the moment, and the size of the rectangle are constructed. To further enhance the robustness of the detector, the size of the rectangle is checked to eliminate false identification as the block. Then after computing the real world coordinate of the blob center, the algorithm also conducts a position check to make sure the objects outside of the table as well as the arm itself will not be detected. Next, the contours of the block are drawn. The color of the contour are chosen to match the color detected so that one can tell if the returned color is correct, which is shown in Figure 3. Finally the center coordinates are returned and stored for Acting section.

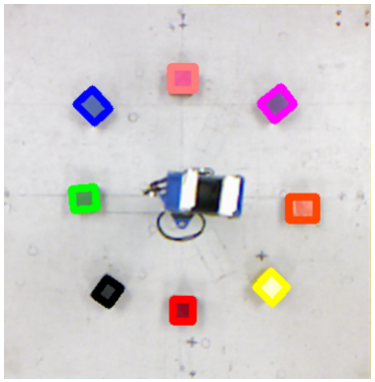


Fig. 3. In our blob detector, the contours of the block are drawn in different color to clearly show if the returned color matches the actual block color

### C. Acting

1) *Gripper Design*

The gripper design was based on the four-bar linkage mechanism, which transformed the rotation of the servo motor into the gripping motion. In order to achieve better performance, another servo motor was used to rotate the whole gripper in its own axial direction. Therefore, the robotic arm used had six degrees of freedom in aggregate. The weight of the gripper was expected to be light, so that it would not add extra load to the arm motors. The dimension of the gripper was determined to be slightly larger than the blocks. The four bar linkages were first created several assembly holes to test different sizes, and then was finalized it as Figure 4.

The main principle of selecting the gripper design was the simplicity and reliability. The amount of parts used were reasonable, and they were all simple and easy to assemble. The final gripper center was in-line with the axial center of the arm. The assembled gripper can be seen in Figure 5.

The main advantages of this gripper design were its small size and thin structures. It could reach small gaps to pick up blocks which were close to each other. The reliable mechanism and sufficient torque provided by the motor

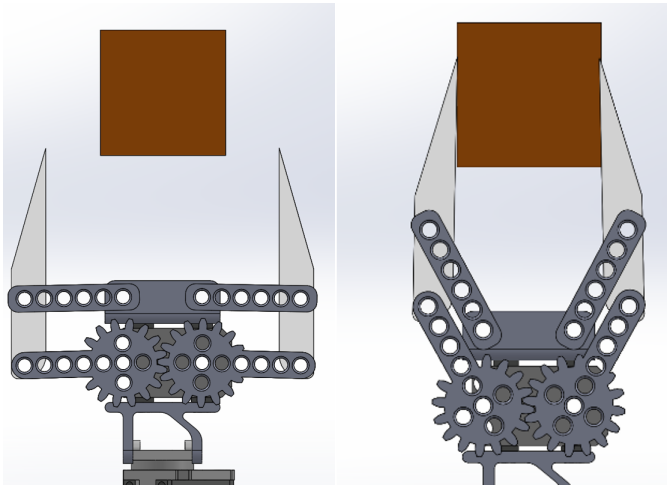


Fig. 4. This figure illustrates the gripping motion with the left one shows the full release state and the right one shows the close state

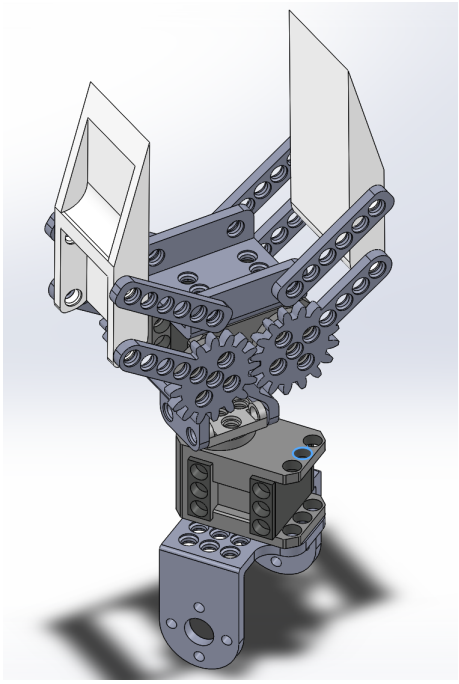


Fig. 5. The 3D rendering of the gripper assembly

also ensured great efficiency for the gripping motion. The additional rotation provided by the lower motor could adjust blocks into the same orientation when putting them down, which significantly helped align blocks in a straight line.

Some disadvantages were also found through the process of this project. The worst one was its physical limitation in space. It was unable to reach some positions with the gripper being horizontal or vertical to the testing plane. Blocks picked up or released at those positions would have an angle between its bottom and the test plane, which made them unstable. In order to deal with this issue, the speed was decreased when picking up or releasing blocks at those positions and it helped solve most of the cases.

2) *Joint Angle Limits*

Each joint in the arm was a rotational joint controlled by a servo motor. Due to the physical limitation of the structure and the length of wires, each joint could only move between certain angles. We first used the feedback from motors to record these angle limits, and then write the clamp function with these values. If any value exceeding the limits was commanded, the clamp function would constrain them to the limits before publishing these command. Table I gave the limits set to each joint. "Gripper 1" is the motor controlling the gripper rotation and "Gripper 2" is the one controlling the gripping motion.

TABLE I

JOINT ANGLE LIMITS

| Base | Shoulder | Elbow |
|---|---|---|
| ±200° | ±124° | ±122° |
| Wrist | Gripper 1 | Gripper 2 |
| ±105° | ±180° | ±60° |

3) *Forward Kinematics*

Forward kinematic was used to get the location and orientation of the gripper based on the feedback joint angles. Devanit-Hartenberg Table was used to do the math. Parameters used in the calculation was shown in Table II. Since the rotation and gripping motion of the gripper would not affect its end position and orientation relative to ground, those two motor angles were included in the gripper. The global frame of reference is shown in Figure 6.

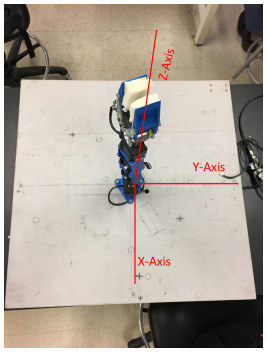



Fig. 6. The global xyz axis shows in the work station with the origins at the base of the arm

TABLE II

DEVANIT-HARTENBERG TABLE

| Link | $\alpha_i$ [rad] | $a_i$[mm] | $d_i$[mm] | $\theta_i$[rad] |
|---|---|---|---|---|
| 1 | 0 | 0 | 120 | 0 |
| 2 | $\pi/2$ | 0 | 0 | $\theta_{base}$ |
| 3 | 0 | 100 | 0 | $\theta_{shooulder} + \pi/2$ |
| 4 | 0 | 100 | 0 | $\theta_{elbow}$ |
| 5 | 0 | 155 | 0 | $\theta_{wrist}$ |

Then, the location of the gripper effective end can be calculated by changing the frame of reference:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = R_0^1 R_1^2 R_2^3 R_3^4 R_4^5 \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \tag{5}$$

$$R_{n-1}^n = \begin{bmatrix} \cos\theta_n & -\sin\theta_n \cos\alpha_n & \sin\theta_n \sin\alpha_n & a_n \cos\theta_n \\ \sin\theta_n & \cos\theta_n \cos\alpha_n & -\cos\theta_n \sin\alpha_n & a_n \sin\theta_n \\ 0 & \sin\alpha_n & \cos\alpha_n & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

$$\phi = \theta_4 - \frac{\pi}{2} + \theta_2 + \theta_3 \tag{7}$$

3) *Inverse Kinematics*

Inverse kinematics was essential to the motion planning. It transformed the position in space into the joint angles of each motor, which enabled us to command the gripper to move to the desired position.

Because of the enough degrees of freedom, the orientation of the gripper was set as an input to the calculation, which means the orientation between the gripper and the ground can be inputs. at each position. Due to the relative simplicity of the joints, the inverse kinematics was determined by purely geometric calculation. Main symbols used in the calculation were explained in Figure 7 and Table III.

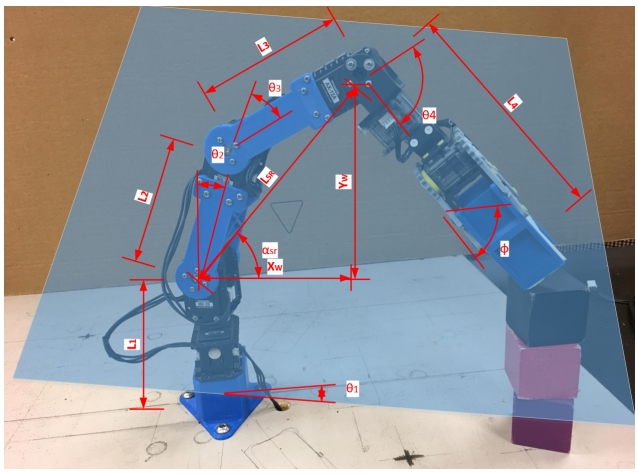



Fig. 7. Inverse kinematics diagram with the blue area shows the projection plane

The first base rotation angle was easy to determined by:

$$x_W = \sqrt{x^2 + y^2} - L_4 \cos(\phi) \tag{8}$$

$$y_W = z - L_4 \sin(\phi) - L_1 \tag{9}$$

$$\alpha_{sr} = \arctan 2(y_w, x_w) \tag{10}$$

$$L_{sr} = \sqrt{x_W^2 + y_W^2} \tag{11}$$

TABLE III

SYMBOLS FOR INVERSE KINEMATICS

| Symbol | Unit | Explanation |
|---|---|---|
| $L_1$ | [mm] | Length of link from ground to the shoulder joint |
| $L_2$ | [mm] | Length of link from shoulder joint to the elbow joint |
| $L_3$ | [mm] | Length of link from elbow joint to the wrist joint |
| $L_4$ | [mm] | Length from wrist joint to effective gripper end |
| $\theta_1$ | [rad] | Rotation angle of base joint motor |
| $\theta_2$ | [rad] | Rotation angle of shoulder joint motor |
| $\theta_3$ | [rad] | Rotation angle of elbow joint motor |
| $\theta_4$ | [rad] | Rotation angle of wrist joint motor |
| $x$ | [mm] | x coordinate in desired position |
| $y$ | [mm] | y coordinate in desired position |
| $z$ | [mm] | z coordinate in desired position |
| $\phi$ | [rad] | Desired gripper orientation relative to ground |

Then, the following calculation can be performed at one single plane indicated by blue in Figure 7 For the elbow upward configuration:

$$\theta_2 = \frac{\pi}{2} - \alpha_{sr} - \arccos\left(\frac{L_2^2 + L_{sr}^2 - L_3^2}{2L_2L_{sr}}\right) \tag{12}$$

$$\theta_3 = \pi - \arccos\left(\frac{L_3^2 + L_2^2 - L_{sr}^2}{2L_2L_3}\right) \tag{13}$$

$$\theta_4 = \phi + \frac{\pi}{2} - \theta_2 - \theta_3 \tag{14}$$

For the elbow downward configuration:

$$\theta_2 = \frac{\pi}{2} - \alpha_{sr} + \arccos\left(\frac{L_2^2 + L_{sr}^2 - L_3^2}{2L_2L_{sr}}\right) \tag{15}$$

$$\theta_3 = -\pi + \arccos\left(\frac{L_3^2 + L_2^2 - L_{sr}^2}{2L_2L_3}\right) \tag{16}$$

$$\theta_4 = \phi + \frac{\pi}{2} - \theta_2 - \theta_3 \tag{17}$$

### D. Reasoning

1) *State Machine*

A moore finite-state machine [4] or deterministic finite state automata is used to describe the planning logic of the algorithm. Though each task has different waypoint plans, the global motion process follows a general state machine. The general state machine is represented in a flowchart, which uses the standard flowchart symbols [5].

Figure 8 shows a comprehensive flowchart of state machine. As the program is running, software detects all blocks positions within the workspace and store the world coordinates under the video class. Once an user inputs a task command to excite the automated task function, the program acquires world coordinates of desired blocks and passes them one by one to the sub-functions to determine waypoints of the motion. Command arm move to grab block contains a for loop algorithm, by taking each waypoint and transforming through configuration space using inverse kinematic method mentioned in the Acting section. It publishes commands to move the robotic arm from the current position to next

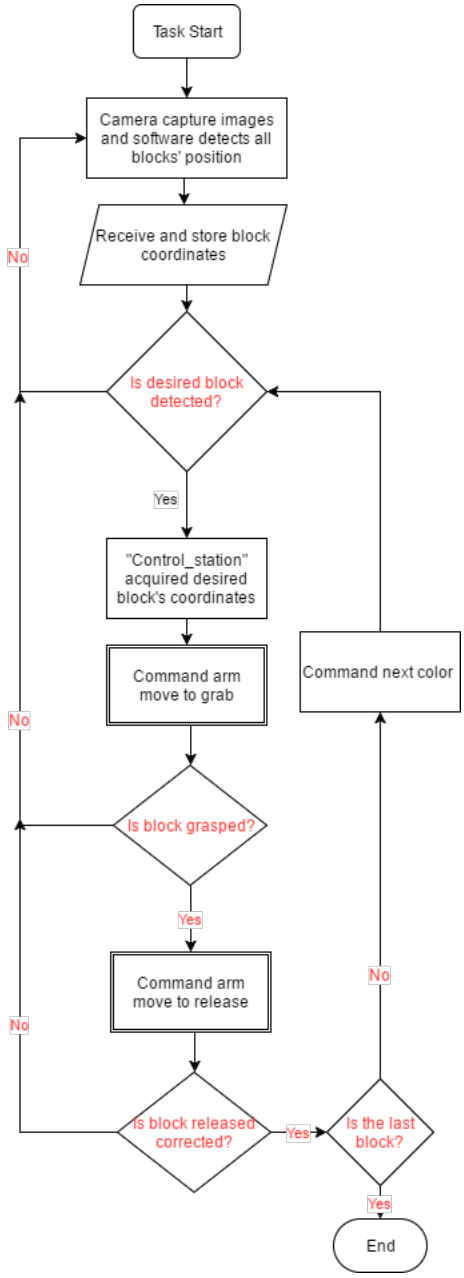



Fig. 8. Full flow diagram of state machine with red color shows decision making process

location. Meanwhile, the LCM feeds the motors' positions to the program, which are used to check if the robotic arm has reached to the next position. After each grasp command, the program checks the motor load feedback to ensure the block is successfully grabbed. Similarly, after a proper release, the program will determine whether the block has reached the final release location by checking the new position of the block. If the new position deviates, the program will roll back to detection and redo the previous steps until it reaches the position within the limit. The state machine keeps running until it reaches the last point/location of the path plan. Next section contains the detail description of the path planning.

2) *Grasp/release logic*

The inverse kinematic requires the $\phi$ and elbow configuration as the inputs. Determination of the $\phi$ and elbow configuration are task-different and each task has its own strategy to choose $\phi$. Whereas, due to the physical setup of the gripper, elbow-up configuration is favored. Meanwhile, each $\phi$ is picked by an iterative process, which finds the first feasible $\phi$ out of the inverse kinematic calculation. The instruction of 5 tasks are summarized in the lab manual. Since $0^o$ and $90^o$ are mostly desired configurations, the first two tasks use the iterative method to find the $\phi$. The first two trial, $phi$, $\phi_0 = 90, \phi_1 = 0$, have determined, and the rest $\phi$ iteration use the equations, 18 and 19 to calculate.

$$\phi_{2i} = 90 - i, i = 1 \cdots k \tag{18}$$

$$\phi_{2i+1} = 0 + i, i = 1 \cdots k \tag{19}$$

Task 1 and Task 2 use the same method to get $\phi$ value and task 3 uses the same method to calculate $\phi$ for grabbing, but while lining up the blocks, task 3 only favors $90^o$ configuration. This release process uses an iteration to decrease $\phi$ from $90 * o$. For task 4 and task 5, the release process prefers close to $90^o$ below 5th level of block stacks and $0^o$ is preferred otherwise. This configuration gives fairly stable stacking availability for this gripper. Towards to the twist/roll angle, task 1, task 2, and task 4 has a fix negative $90^o$ orientation. Task 3 and task 5 are required to place blocks in a same orientation so that the twist angle is defined by 20:

$$\theta_4 = \theta_0 \tag{20}$$

By releasing the block as equation 20, the blocks orientations are parallel to each others.

3) *Plan Algorithm*

For every grab motion step, it contains a iterative process as shown as Figure 9. Release motion has a similar iterative mechanism to finish the operation successfully. The grasp-plan algorithm takes the blocks world coordinate as the input and add intermediate points in order to move freely to the top of the block without hitting any block. The gripper is commanded to move to projected location on 2D space, but it is around 100 - 150 mm higher than the block center as a pause. During this movement, the arm is commanded to operate at a high speed mode to speed up the process. Once it reaches the place, the program will switch it to a low speed mode to to increase the success rate of the execution. For release process, program has similar waypoints plan as the grab but operating every step in a low speed command to keep holding block stable.

To improve the smoothness of each steps, a cubic spline method can be used. With cubic polynomial trajectories, each joint starts at an unified speed and reached to final orientation at the same time. The general cubic spline equations [6] are:

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \tag{21}$$

$$\dot{q}(t) = a_1 + 2a_2 t + 3a_3 t^3 \tag{22}$$

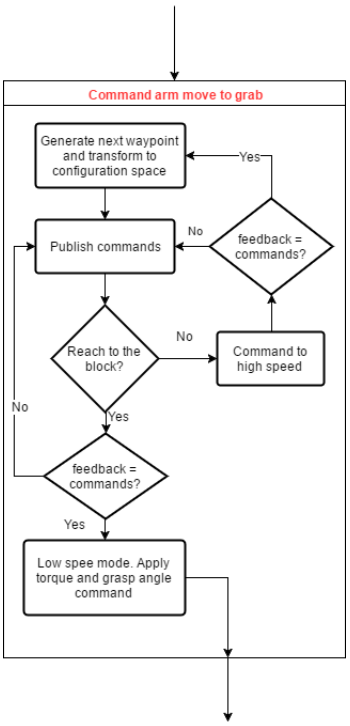



Fig. 9. Flow diagram of move to grab

After applying four boundary conditions, those equations can be written as the following linear system:

$$\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 \\ 0 & 1 & 2t_0 & 3t_0^2 \\ 1 & t_f & t_f^2 & t_f^3 \\ 0 & 1 & 2t_f & 3t_f^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} q_0 \\ v_0 \\ q_f \\ v_f \end{bmatrix} \tag{23}$$

The cubic spline algorithm solves this matrix using least square method from numpy module [2] to get the four coefficients, $a_0$,$a_1$,$a_2$,and $a_3$. To solve those four coefficients, four boundary has to be defined first. $q_0$ is the initial displacement/rotation, $v_0$ is the initial velocity, $q_f$ is the final displacement/rotation, and $v_f$ is the final velocity. Rather than using cubic spline method, dynamically choosing waypoints are also used for those tasks. The inverse kinematic method is implemented during the path planning. The world location of the motion is in the last of the configuration chain and the geometric inverse kinematic method can return the joint angles, which are used as the command signals. Different from picking up blocks and release them in final location, task 2 and task 4 have add a middle location. Task 2 release the grabbed block in a closed-middle location and grasp it with a $90^o$ configuration, which increase the success rate of stacking it. Task 4 have the similar way points plan, but additional to task 2's plan, task 4 include an $90^o$ twist before it re-picks up the blocks. This additional waypoints and operations increase the smoothness contact between the gripper and blocks. Lastly, each task has to determine the order of pickup blocks and design path to avoid collision. From the color aspects, the coordinates are stored in an order of the resistor color code (except the last one is pink). Task

1 and Task 2 keeps the pickup order as the color stored order. Task 3 has one more priority before using the color code. The robotic arm comes to pick up the blocks at the highest level first in a resistor color order and place them in their corresponding locations. Different to previous tasks, task 4 has to stack blocks in a resistor color order so that the program has to define few safe spots to relocate blocks. The program chooses to move high-level blocks to those safe spots until it capture see all eight blocks. Once all eight blocks have been seen, the program commands to pick up blocks in the resistor order. For task 5, the determination is more freely, as long as it can detect blocks within the top $3/4$ section of the workspace, it will pick the block and place to the next location of the pyramid. The order of the final location of the pyramid has shown on Figure 10.

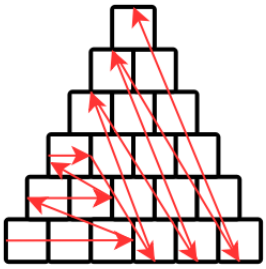


Fig. 10. The stacking order of pyramid is shown. The 3 level pyramid is completed first, and then complete the rest of by moving diagonally

Currently, if the rexarm does have the potential of colliding, the program will tell the robot to move at a higher vertical level. This method has been successfully compatible with all tasks since the highest potential height is 3 blocks stack. For task 5, once it reaches to the fourth floor of the pyramid, the arm will always rise to the highest position during the delivering to avoid collision. Overall, the perfomance has reached to the top limit of our gripper and speed design. The physical limitation of the gripper constrains the accessibility to a certain region and pick up the block vertically downward. Inconsistent between pick up angle, $\phi_{pick}$, and release angle, $\phi_{release}$ introduces potential risk of tipping and rolling. To prevent tipping, gripper suppose to have a same angle, $\phi$, for picking and releasing. By accomplishing new gripper design, the robotic arm could reach further location and process motion smoothly. The system currently does not use high level collision check algorithm. Collision are avoided by the intuitive operating. A potential field [7] and probabilistic Roadmap path planning [8]. A potential field generates a 2D or 3D vector field for object motion planning and it automatically use the gradient of a potential function to create obstacles free path. However, as Dr.Kuipers [7] mentioned, potential fields have possibilities of converging to a local minima and generating unstable oscillation. The computation requirement has beyond the scope of this project and the potential issues are sophisticated. Another option of improving collision checking is to implement probabilistic roadmap path [8]. By determining the geometric obstacles, the planner could generate a free path for manipulator to move from initial position to the final position. The robust algorithm handles additional situations rather than the current task and potential cases. Similar to the potential field, the complexity has exceeded the scope and simple method have successful solved all the collision issues.

## III. RESULTS

### A. Teach and Repeat

The robotic arm had successfully finished the teach and repeat task. The waypoints are chosen to poke the holes without contacting with the edge of holes. By adding extra waypoints on the top of each hole before poking it increased the success rate. 11 is a waypoint trace presented in world coordinates of a successful trial. Extensive middle points highly increased probability of completing tasks for both low speed and high speed models. However, high speed model had a higher accuracy by completing task eight times out of eleven times and low speed model had pass rate, seven eleventh. Overall, as the number of waypoints increased, the success rate increases and the difference between two models decreases.

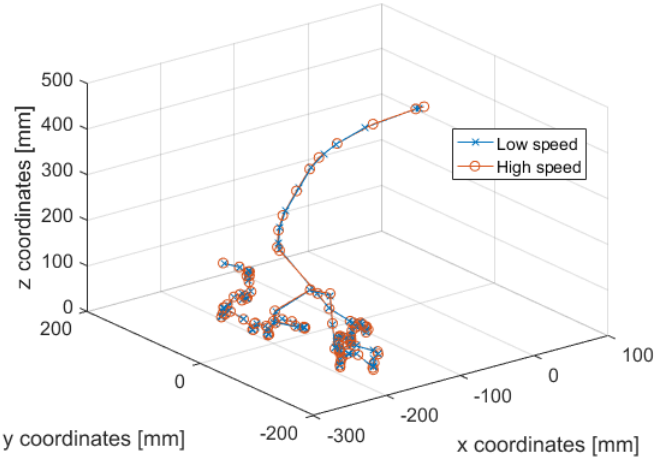



Fig. 11. The trace of arm tip travel under two speed models

### B. Motion Accuracy

The accuracy of the robotic arm motion was tested at several different positions. Those positions were accurately marked by caliper and used as the input of Inverse Kinematics. The arm was commanded to the output angles of Inverse Kinematics. The results calculated by Forward Kinematics based on motor angle feedbacks were also recorded. Then, the gripper was commanded to pick up a block at that marked position.

The difference between the actual gripper position and the commanded position was recorded, so was the difference between the result of Forward Kinematics and the commanded position. Each point was tested for ten times. The testing results was presented in Table IV.

TABLE IV
MOTION ACCURACY TEST RESULT

| Position | Coordinate | FK Error | IK Error | Gripping Success |
|---|---|---|---|---|
| 1 | (100,0,39) | 0-1 mm | 0-5 mm | 100% |
| 2 | (-100,0,39) | 0-2 mm | 0-5 mm | 100% |
| 3 | (0,200,39) | 0-1 mm | 2-6 mm | 100% |
| 4 | (0,-200,39) | 0-1 mm | 3-6 mm | 100% |
| 5 | (300,0,39) | 0-2 mm | 3-9 mm | 100% |
| 6 | (-300,0,39) | 0-3 mm | 5-11 mm | 100% |
| 3 | (200,0,78) | 0-1 mm | 7-13 mm | 90% |
| 4 | (-200,0,78) | 0-2 mm | 3-10 mm | 100% |
| 3 | (0,200,117) | 0-2 mm | 5-9 mm | 100% |
| 4 | (0,-200,117) | 0-1 mm | 3-8 mm | 90% |

### C. Competition

After massive tuning and control, the robotic arm has completed task 1 with a success rate over $95\%$, and it failed to complete the task once, while the blocks were placed out of the range. Task 1 completed within 20 seconds during the competition and blocks were placed in correct place with high accuracy. By adding middle points to way-point plan for task 2 operation, the overall operating speed decreased, but the accuracy and completion increased to over $90\%$. During the competition, task2 was completed within 40 seconds, but the three block stacked smoothly.

Task 3 placed eight block in a resistor color order and due to the gripper physical limit, blocks that were far away from the center was not placed parallel to others. However, the program added extra motions to use the advantage of the gripper design and pushed the two blocks from outside to the center. The outcome of task 3 was excellent and the exact performance can be viewed from "team7_task3_1" video. The overall success rate of this task is over $90\%$. Task 4 was the weakest task for this gripper and program at the beginning, but adding a middle point and a $90^o$ twist operation boosted the performance of this task. During the competition, task 4 was completed slowly but with a higher success rate and it only took one trial to finish the task. Lastly, a 6 level pyramid was completed during the competition in the second trial, even though it was not completed at the first time. The video was not recorded during the competition, but an early trial was recorded and uploaded to the same folder, named as "team7_task5_2". The algorithm does have a completed rate over $90\%$ for five-level stacking, but a lower rate for six-level pyramid.

TABLE V
TASK PERFORMANCE

| Event | Time(s) | Points | Performance |
|---|---|---|---|
| 1 | 19.5 | 97.5 | Good |
| 2 | 34.7 | 92.5 | Good |
| 3 | $1'27$ | 97.5 | Second block to the left is a little off |
| 4 | $2'29$ | 120 | Fourth block is a little off |
| 5 | 4.40 | 250 | Stack to 6 level |

## IV. CONCLUSIONS AND FUTURE WORK

In conclusion,all the tasks were successfully completed in this project, including the gripper design, camera calibration, block detection, forward and inverse kinematics, state machine planning and all the five competition tasks. The performance of this prototype in the competition was great, ending up with the second highest score. The entire system worked well after solved different kinds of problems and overcame the physical limitation of the system as much as possible.

There were few improvement could be conducted to push the performance. The method used to detect blocks outputs four vortex points and all four points are used to define the center of block. However, the orientation is not one of the outputs, which could improve the performance in real application. With the correct orientation, the block can be easily picked. Due to the perception requirement, the block orientation determines the pickup quality, potential failure during delivery and placement success rate.

The programming have revealed all potential of the current gripper design, but clearly not all of tasks have completed with a high success rate. The robotic arm has programmed for checking if it picked up block successfully for most times. However, during the last pyramid task, a proper release check has not been added to the state machine process. Strong overlapping of blocks bring difficulties of distinguish block color and height for this task. A failure of return block position will cause the release check function to be not successful. In the future, a better release function check should be added to the program in order to keep the state machine in a full state feedback rather than a conditional state feedback for certain tasks.The collision check could also be improved by implement other methods to make the system to be more robust.

## REFERENCES


[1] R. Sharpe, *Affine Differential Geometry: Geometry of Affine Immersions*. 521 Cambridge University Press, 1997.

[2] (2017) Numpy for matlab users. [Online]. Available: https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html

[3] Geometric image transformations.

[4] tutorialspoint.com, "Moore and mealy machines." [Online]. Available: https://www.tutorialspoint.com/automata_theory/moore_and_mealy_machines.htm

[5] Flowchart symbols meaning — standard flowchart symbol images and usage. [Online]. Available: https://creately.com/diagram-type/objects/flowchart

[6] M. W. SPONG, *ROBOT MODELING AND CONTROL*. JOHN WILEY & Sons, 2012.

[7] B. Kuipers. Potential fields and model predictive control. [Online]. Available: http://www.cs.utexas.edu/~pstone/Courses/395Tfall05/resources/week9b-ben-potential-fields.ppt

[8] H. M. Choset, *Principles of robot motion theory, algorithms, and implementation*. MIT Press, 2005.

## APPENDIX

### A. Gripper Bill of Materials (BOM)

All parts are shown in Figure 12.Lists of the parts are shown in Table VI.Details of each part can be found by engineering drawings attached at the end of this report.

TABLE VI
BOM OF GRIPPER

| Part | Quantity | Material/Type |
|---|---|---|
| Bar | 4 | ABS |
| Clamp | 2 | ABS |
| Gear Link 1 | 2 | ABS |
| Gear Link 2 | 2 | ABS |
| Gripping Motor Hold | 1 | ABS |
| Gripping Motor Top | 1 | ABS |
| Side Support | 1 | ABS |
| Wheel | 5 | ABS |
| Servo Motor | 2 | XL320 |

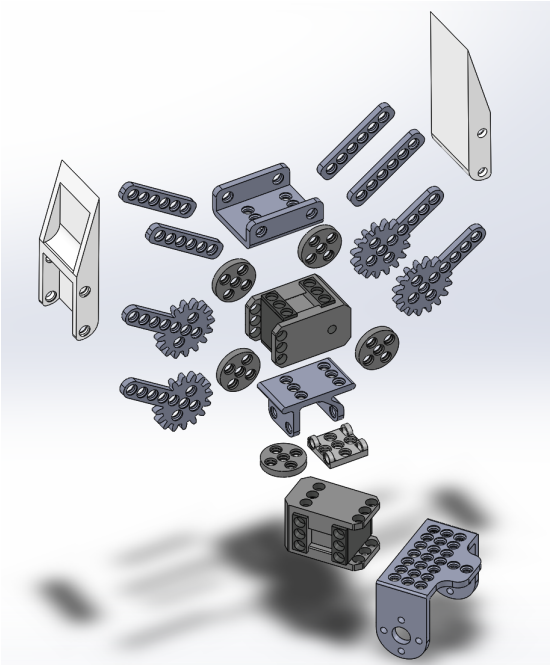


Fig. 12. Gripper BOM

### B. Assembly Instruction

1. Connect the two servo motors and make sure they are at initial status (angle = 0).
2. Attach five circular wheels on motors, four on gripping motor and one on rotating motor.
3. Build the entire gripper from bottom to top.
4. Fix the gears such that each pair is horizontal.
5. Insert the two clamps between gear links and bars.

"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 02/17/2017."

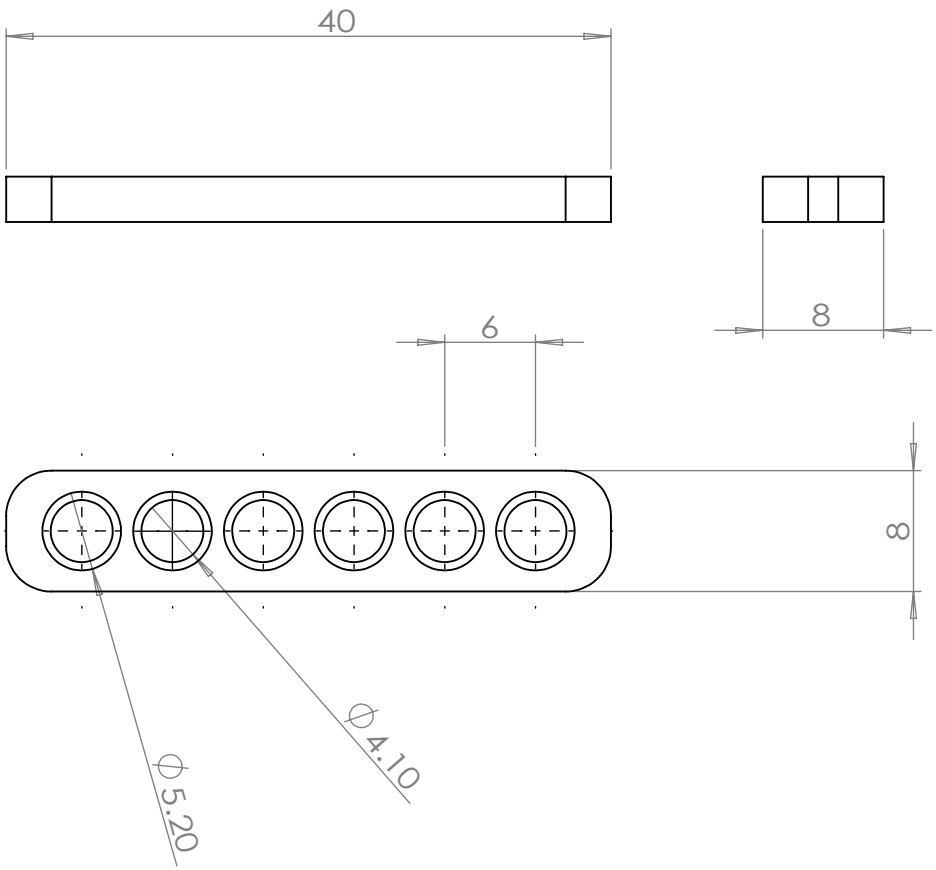
2/17/2017

"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 02/17/2017."

2/17/2017

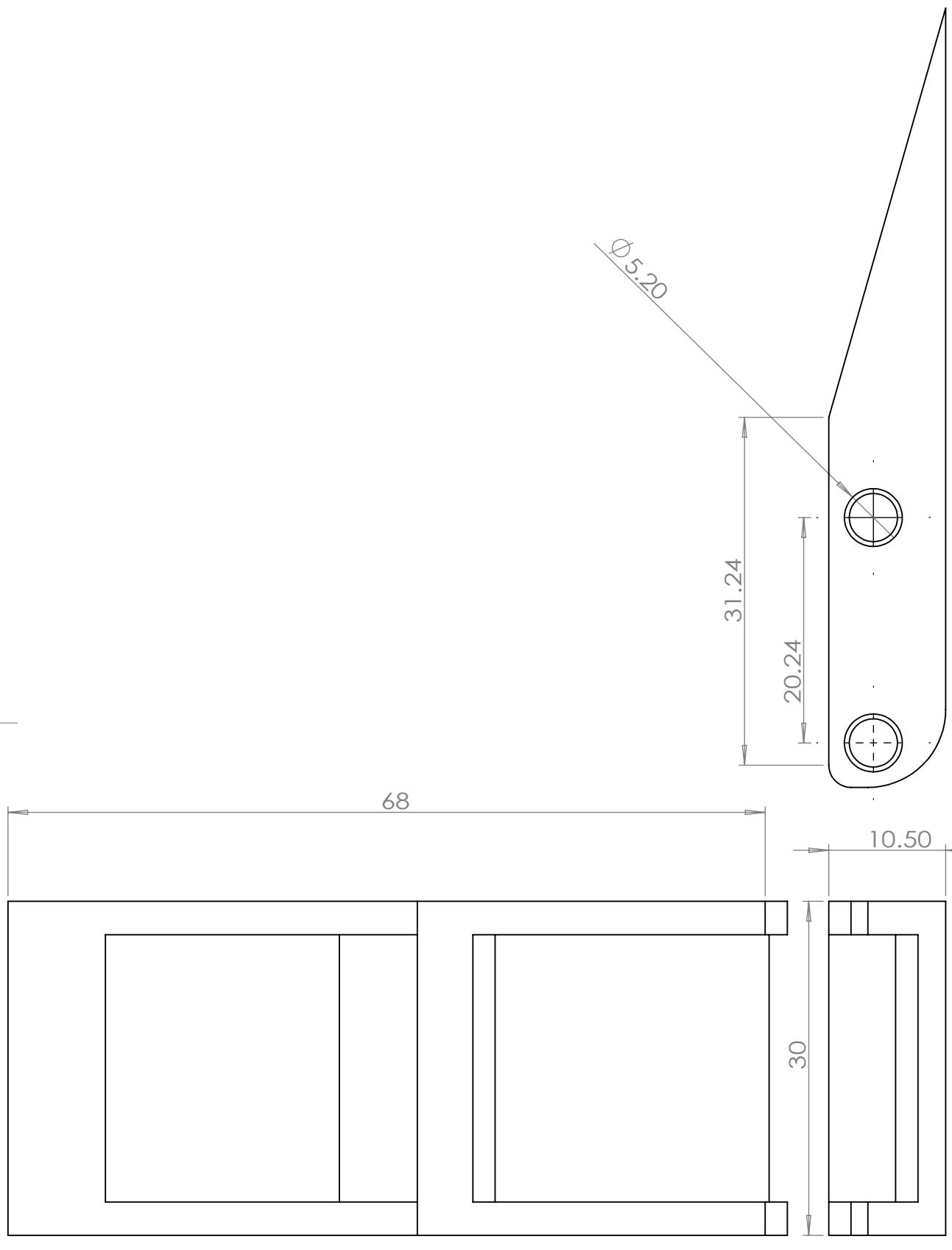
"I participated and contributed to team discussions on each problem, and I attest to the integrity of each solution. Our team met as a group on 02/17/2017."

2/17/2017



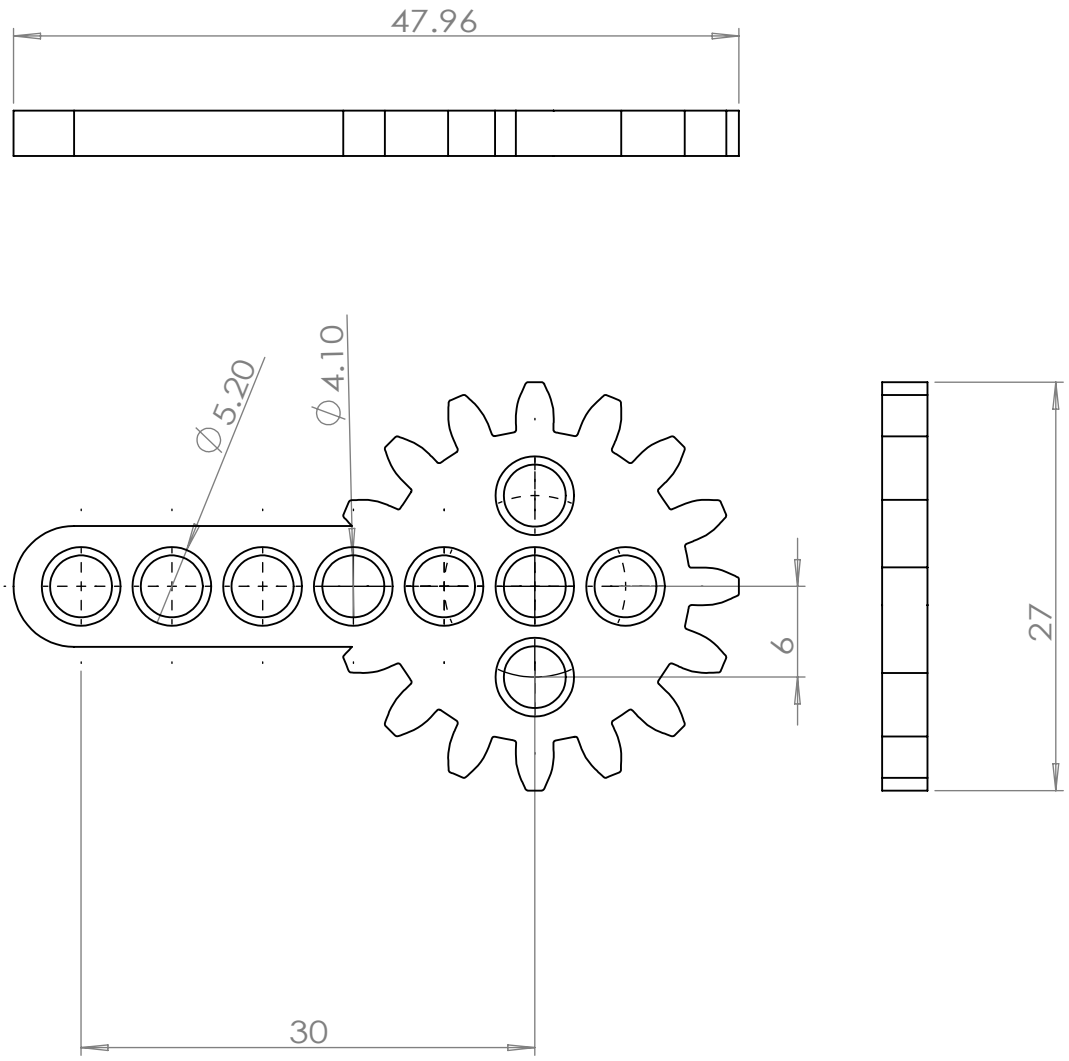

**SOLIDWORKS Educational Product. For Instructional Use Only**

| | | |
|---|---|---|
| TITLE: ROB 550 ROBOTIC ARM | | |
| DWG NO. bar | | TEAM 7 |
| SCALE:2:1 | UNIT: mm | SHEET 1 OF 8 |



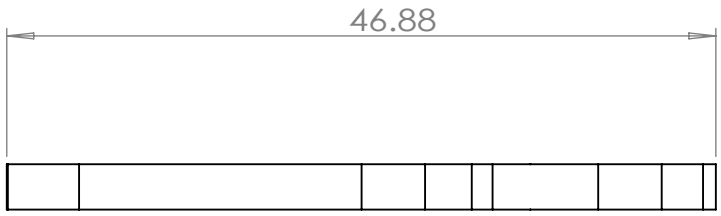

| | | |
|---|---|---|
| TITLE: ROB 550 ROBOTIC ARM | | |
| DWG NO. clamp_long | | TEAM 7 |
| SCALE:2:1 | UNIT: mm | SHEET 2 OF 8 |

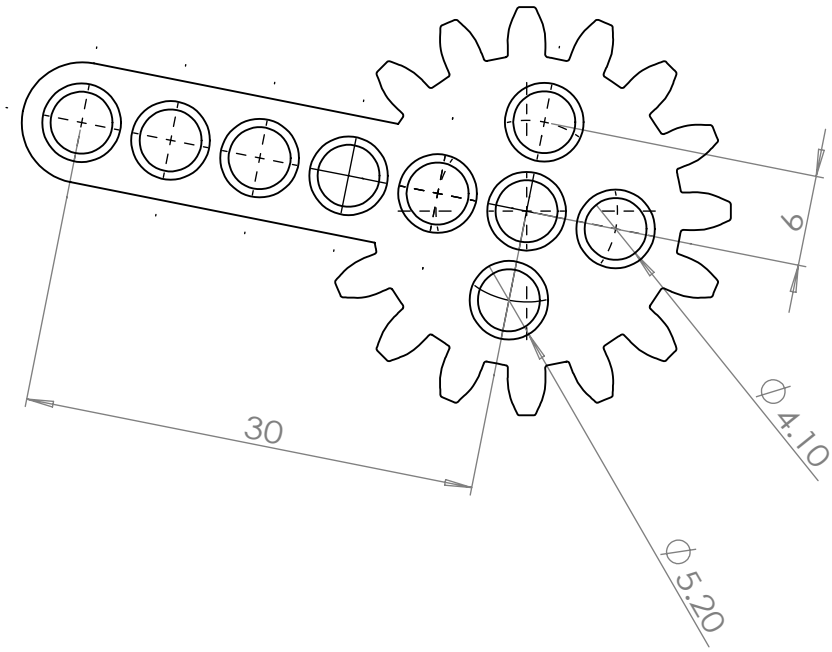
**SOLIDWORKS Educational Product. For Instructional Use Only**

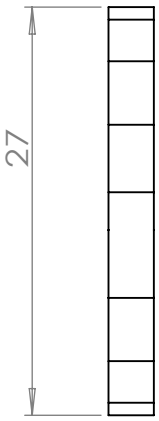



**SOLIDWORKS Educational Product. For Instructional Use Only**

| | | |
|---|---|---|
| TITLE: ROB 550 ROBOTIC ARM | | |
| DWG NO. gear_link1 | | TEAM 7 |
| SCALE:2:1 | UNIT: mm | SHEET 3 OF 8 |



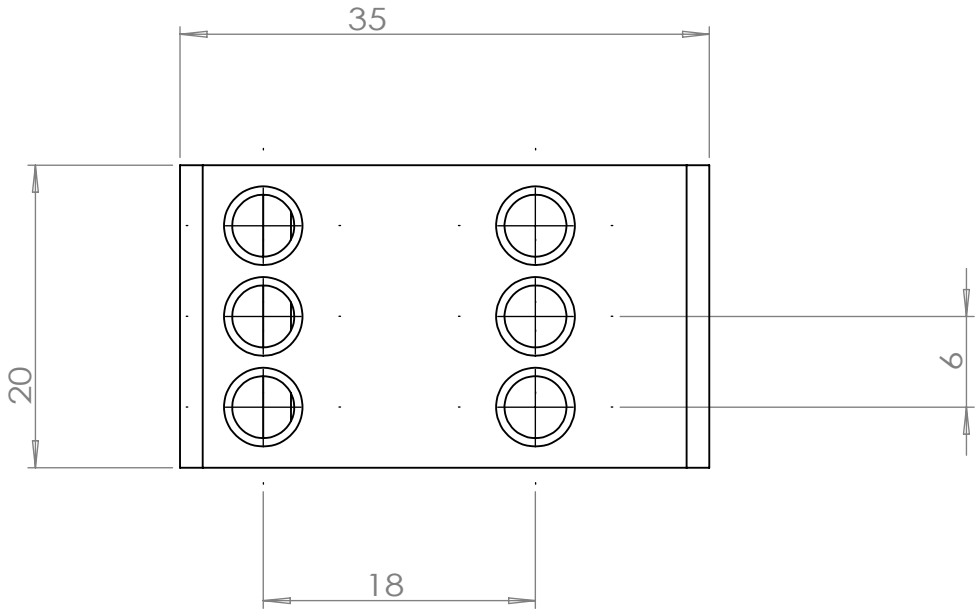



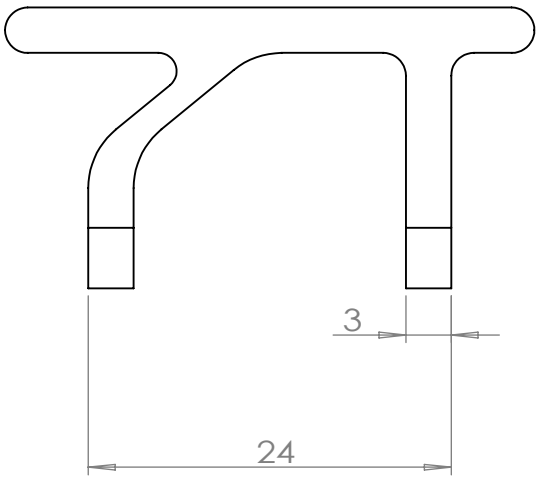

**SOLIDWORKS Educational Product. For Instructional Use Only**

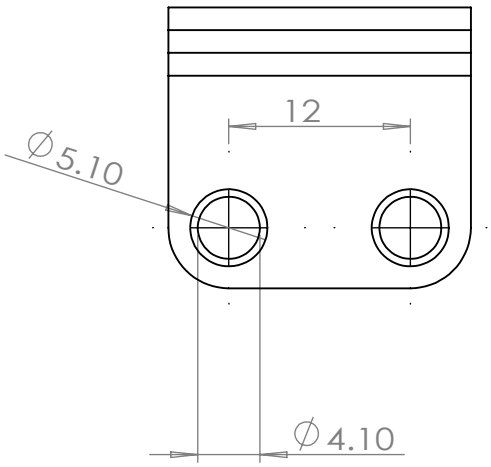
TITLE:

# ROB 550 ROBOTIC ARM

| DWG NO. | | TEAM |
|---|---|---|
| gear_link2 | | 7 |
| SCALE:2:1 | UNIT: mm | SHEET 4 OF 8 |



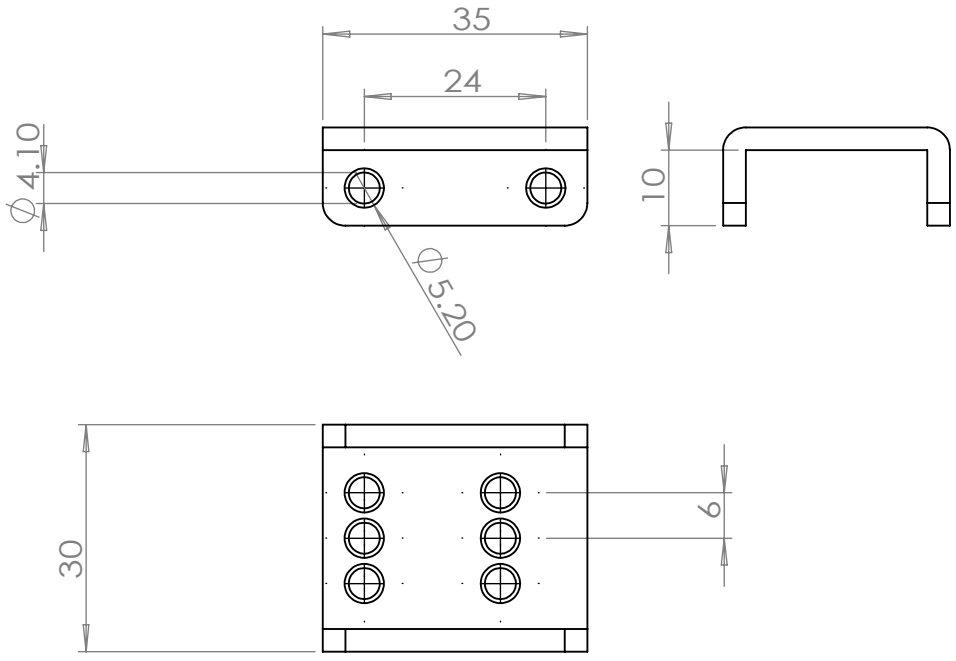

**SOLIDWORKS Educational Product. For Instructional Use Only**

TITLE:

# ROB 550 ROBOTIC ARM

| DWG NO. | TEAM |
|---|---|
| gripping motor hold | 7 |

| SCALE:2:1 | UNIT: mm | SHEET 5 OF 8 |
|---|---|---|



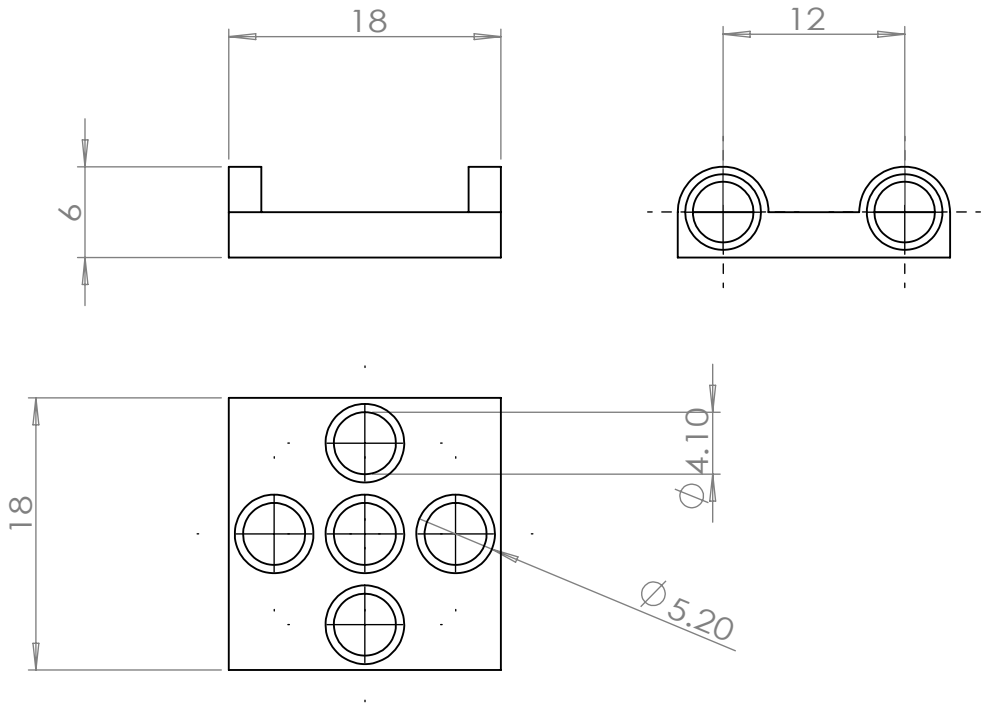

**SOLIDWORKS Educational Product. For Instructional Use Only**

TITLE:

# ROB 550 ROBOTIC ARM

| DWG NO. | TEAM |
|---|---|
| gripping motor top | 7 |

| SCALE:1:1 | UNIT: mm | SHEET 6 OF 8 |
|---|---|---|

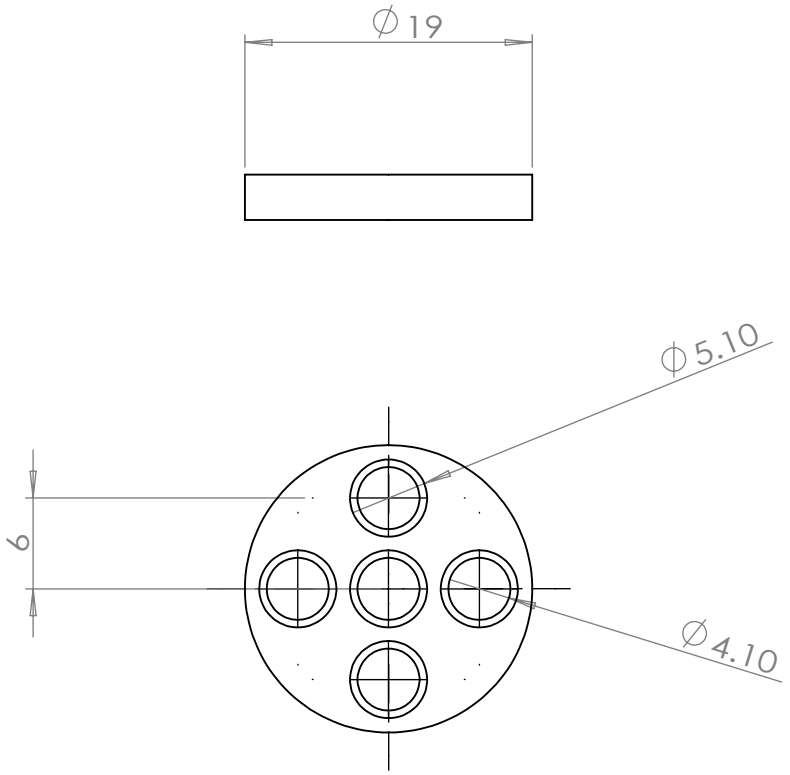



**SOLIDWORKS Educational Product. For Instructional Use Only**

TITLE:

# ROB 550 ROBOTIC ARM

| DWG NO. | | TEAM |
|---|---|---|
| side support | | 7 |
| SCALE:2:1 | UNIT: mm | SHEET 7 OF 8 |




**SOLIDWORKS Educational Product. For Instructional Use Only**

| | | |
|---|---|---|
| TITLE: ROB 550 ROBOTIC ARM | | |
| DWG NO. wheel | | TEAM 7 |
| SCALE:2:1 | UNIT: mm | SHEET 8 OF 8 |