# Abstract

The objective of this project was to prototype and program a mechanism to sort lego bricks into different colours and shapes. This was mainly done through the use of the Lego technics kit, ev shield, arduino uno, and nxt camera. Various designs and ideas such as two axis scissor arm, as well as a three axis arm and were investigated, however a flipper was decided upon as the final design due to its simplicity and effectiveness at fulfilling the objective. Due to various reasons, such as time constraints, lack of opportunity to test with the actual conveyor, and technical difficulties, the team was unable to successfully accomplish the task stated above. These difficulties will be discussed later within the report.

# Table of Contents

# 1. Problem Definition

## 1.1 Characteristics

This project's main objective was to design and program a pick and place robot that could sort four different "Lego" bricks into beer cups that were placed within the last one foot of the conveyor.[1] These bricks were to be placed by the teaching assistant in random order and orientation at the start of the conveyor, however the bumps would always face up.

## 1.2 Requirements

Requirements of this project were as follows: the robot must be able to attach to the platform, as well as be able to distinguish between different sized lego blocks, be able to distinguish between different coloured lego blocks, and sort the lego blocks according to its size and/or shape. Furthermore a requirement due to the implementation of the final design was that the brick must be moved to the center of the conveyor in order to allow for a more accurate data sample to be taken by the camera.

## 1.3 Constraints

Constraints of this project were as follows; only the lego kit with select sensors were allowed to be used to the build of the project. The only other items that were allowed to be added were an arduino uno and sensors that cost less than $20. Furthermore, cups were only allowed to be placed within the last 1 foot of the conveyor, and no cups were allowed to be at the end of the conveyor. The first 1 foot of the conveyor was to be kept clear, and sensors must be put within the 2 foot middle section of the conveyor. One the cups were placed in position, they could not be moved during the duration of the project trial.

## 1.4 Metrics

Metrics were predefined at the start of the project by the team and worked towards as goals. These metrics included being able to properly detect the correct colour, size, and shape of the lego brick, having a gripper capable of gripping the lego from any orientation, having an arm that could extend or traverse the entire length of the end of the conveyor, and having a program running that would be able to use all previous metrics assuming they are working to perform the given task of sorting the lego bricks into their respective beer cups.

# 2. Design Solutions

There were two solutions made to satisfy the “Automated Pick and Place Machine” project design requirements for both the modified and original versions. The final design solution is a single motor automated sorter and is referred to as STCK. This design satisfies the modified version of the project to create a simplified sorting system. The other possible design solution created using three motors to  follow the instructions to make a pick and place machine. The pick and place machine design was nicknamed LMPY.

## 2.1 Final Design Solution: Automated Sorter Machine (STCK)

The automated sorter machine was the second design created for the project. It utilizes a long mechanical arm, one servo motor and the EVShield microcontroller, as seen on Figure 2.1. This design was made as a sorter rather than a pick and place machine. The simple mechanism was hypothesized to work better, than its predecessors, LMPY, which had issues that will be discussed later in the report.

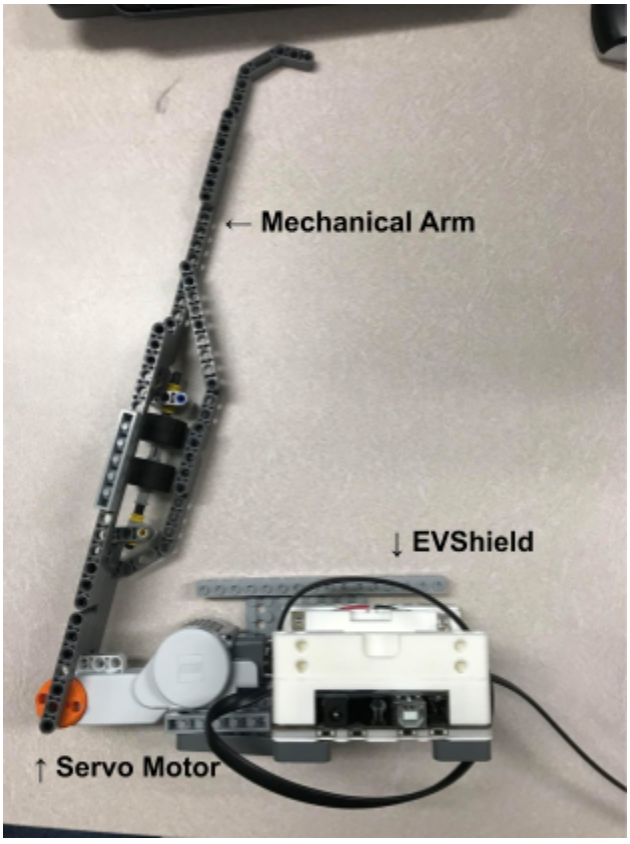


Figure 2.1 STCK Mechanical Components Design

The mechanical arm is connected to directly to the servo motor. The EVShield sends a signal to the servo motor to control its rotational position.Depending on the orientation of the cup, the

EVShield instructs the servo motor to position the mechanical arm to its designated storage location. The motion of the conveyor causes the brick to “roll off” the STCK mechanical arm, as seen in Figure 2.2. When the brick reaches the hooked park of the arm, it falls into the cup placed on the opposite end of the conveyor belt.

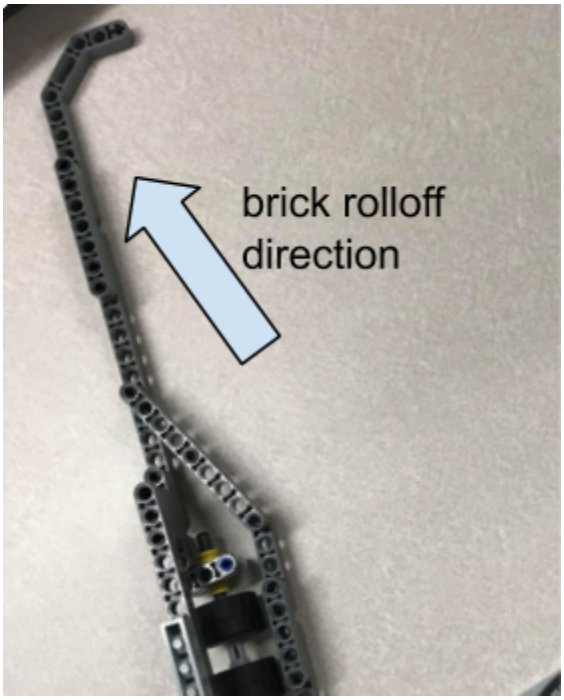


Figure 2.2 STCK Mechanical Arm

A NXT Camera captures the orientation of the brick moving down the conveyor belt. The information read by the NXT Camera is used as an input parameter for the EVShield. The EVShield then uses this information to output a signal to the servo motor, and as stated, controls the motor to position the arm to a designated storage location. A brick positioning mechanism was created before the NXT camera to have the brick positioned in a general area to be read by the camera and directed by the mechanical arm. The NXT camera and brick positioning mount of the STCK is shown in Figure 2.3.

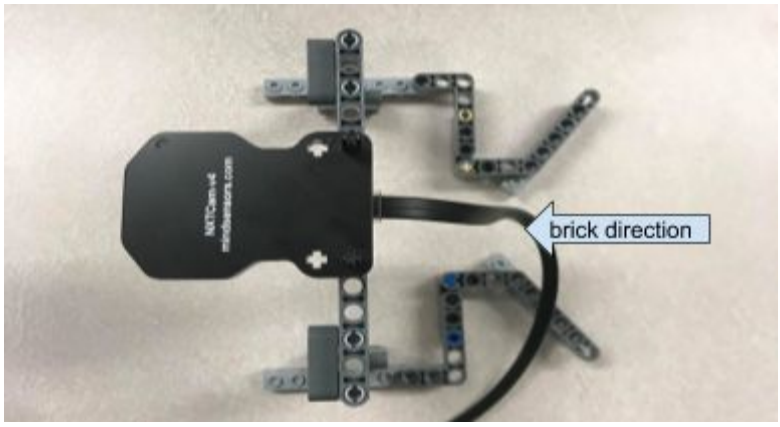


Figure 2.3 NXT Camera and Brick Positioning Mount

## 2.2 Possible Solution: Automated Pick and Place Machine (LMPY)

This design satisfies the original project objective of creating a pick and place machine. LMPY has three servo motors to individually control the claw, the arm extension, and the base of the machine design. The NXT Camera and brick positioning mount, shown in figure 2.3, was originally used and created for this design. As previously stated, the input information for the NXT Camera is used by the EVShield to position the three servo motors to a desired location. FIgure 2.4, shows the starting position of the mechanism while it waits for a brick to enter the mouth of the claw.

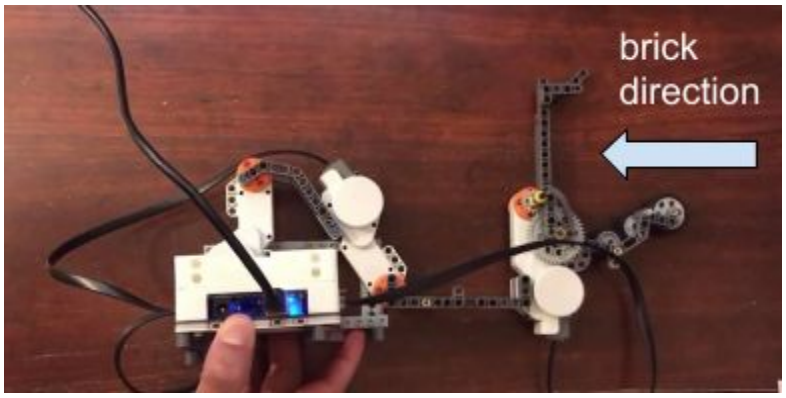


Figure 2.4 Starting position of the LMPY design.

Once the brick is positioned inside the mouth of the claw, the motor of the claw instructed by the EVShield to close, as seen in figure 2.5.

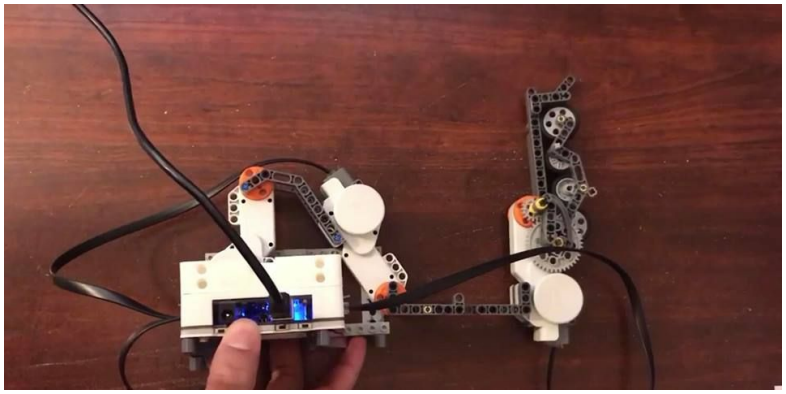

Figure 2.5 LMPY's claw in closed position.

The base motor and the arm extension is then instructed to position the arm to the designated storage location. Figure 2.6 and Figure 2.7 shows the full range extension on the opposite and adjacent side of the conveyor belt, respectively.

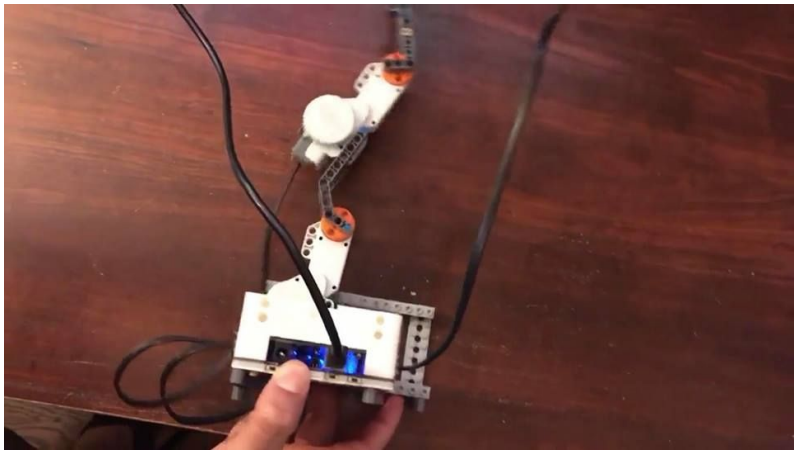

Figure 2.6 LMPY's full range extension on the opposite side of the conveyor belt.

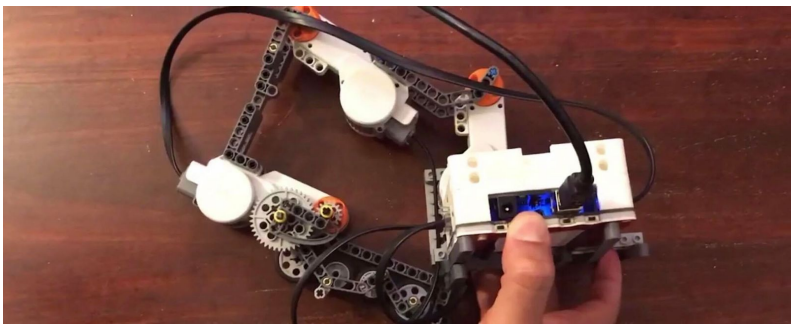

Figure 2.7 LMPY's full range extension on the adjacent side of the conveyor belt.

Once the brick is at its respective location, the claw of the mouth is instructed to open and place the brick into the cup, as shown in Figure 2.8. The mouth of the claw would hover over designated location and the brick would simple fall into cup.

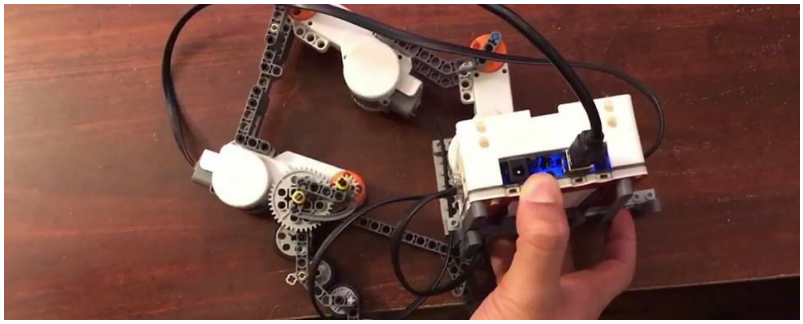

Figure 2.8 LMPY's claw in open position

.

The EVShield then instructs the motors to return to its original position, and is ready for the next brick to come down the conveyor belt, as shown in Figure 2.9.

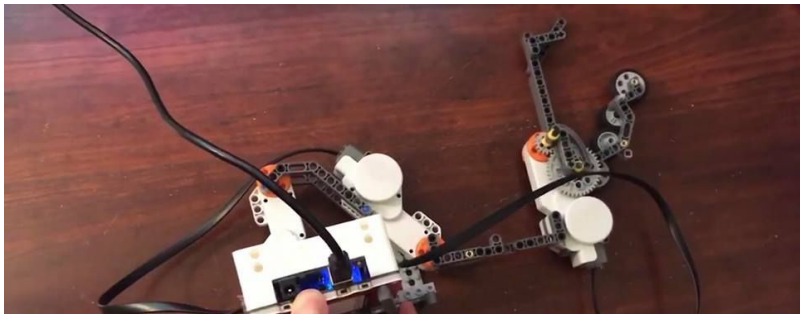

Figure 2.9 LMPY Positioned back to its original catching position

# 3. Black Box Model


Figure 3.1 Black box diagram of the project

The black box diagram shows an overview of all the inputs going into the project system and the desired output from the system. As can be seen from Figure 3.1, the only major input was the constant power supply and the initial code for the sorting system. The system would then take random lego blocks and sort them into their designated cups.

# 4. Simulink model/Sim-Mechanics Model

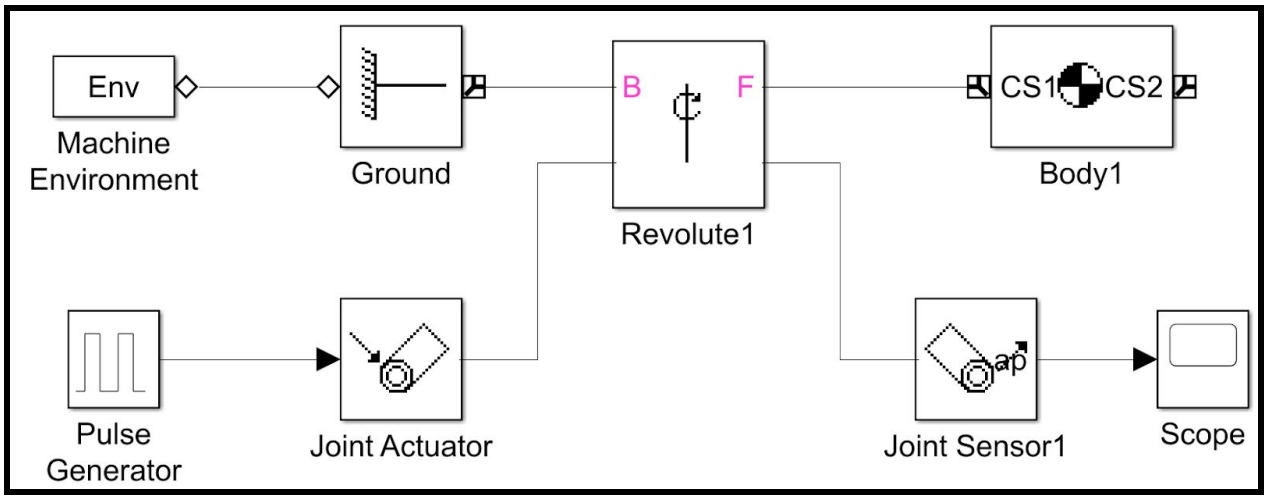


Figure 4.1 Block Diagram of the Sim-Mechanics Model for the STICK Design

In the SimMechanics Model, the robot arm is modelled as one body connected to a revolute joint, which is then connected to ground. The joint is actuated by a PWM generator to simulate the actuation of the lego motors. To monitor the movement of this joint, a joint sensor is connected the joint and the result is displayed in a scope to the experimenter.

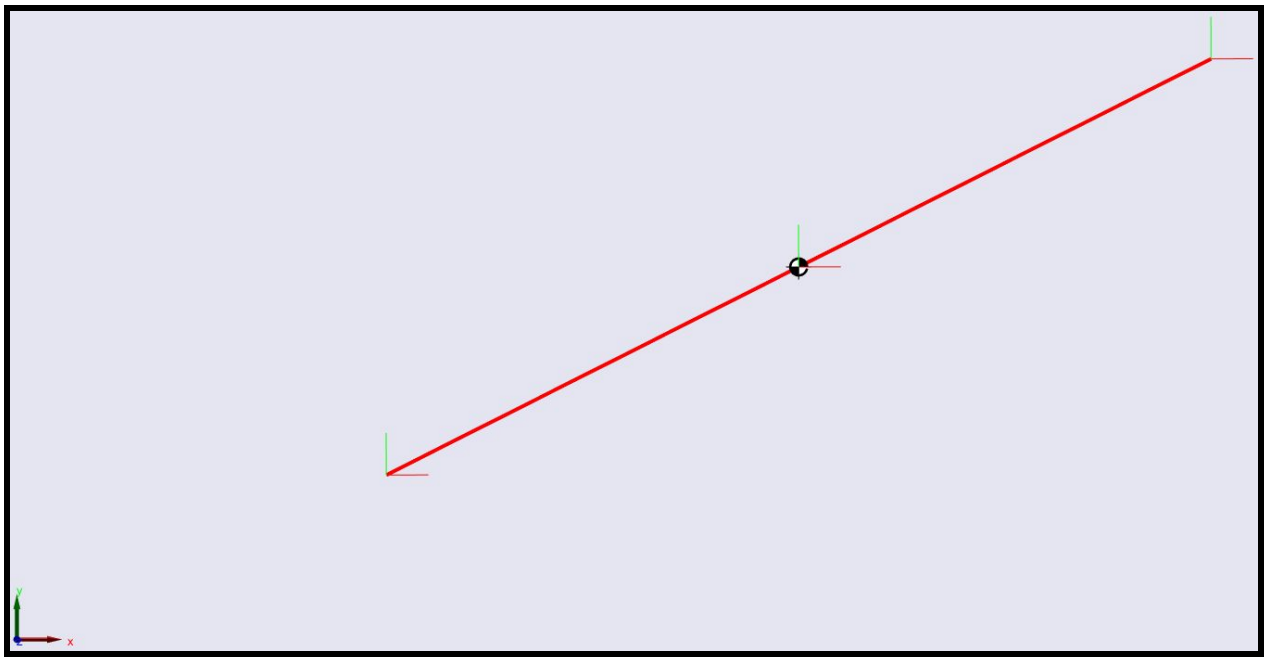

Figure 4.2 Sim-Mechanics Model for the STICK Design

# 5. Discussion of mechanical, electrical, and software implementation

## 5.1 Mechanical implementation

The mechanism built employed one motor from the provided Lego kit. The motor had had more than required torque to move the arm that was attached to the motor. The speed setting used during the arm movement was set at 25. The arm, as a result, was fast in its movement. Once a movement was done, the motor was programmed to use brake hold to hold the current position. This would ensure that the desired angle was reached by the arm in order to guide the bricks into their respective cups. Once a set delay has elapsed, the arm would proceed to move back to its original location.

This motor was secured to the platform beside the conveyor. It was secured in two axis both horizontal) and the EVShield was mounted on top of the motor as an added weight on top of the motor. The EVShield was attached to the same mount as the motor, providing additional structural rigidity and stability.

On the mechanical arm, two wheels were added at the midpoint. These wheels spun as the conveyor belt ran underneath to allow the arm to stay stationary in place. As the arm moved due to motor action, the wheels ensured that the arm height was constant throughout the movement and also during and after the arm went back to home position.

The tip of the arm was curved to allow the bricks to slowly slide along the arm and out into the cups.

## 5.2 Electrical implementation

The EVShield was powered using USB bus power from a computer. The motor and the NXT camera received power from the EVShield via cables.

## 5.3 Software implementation

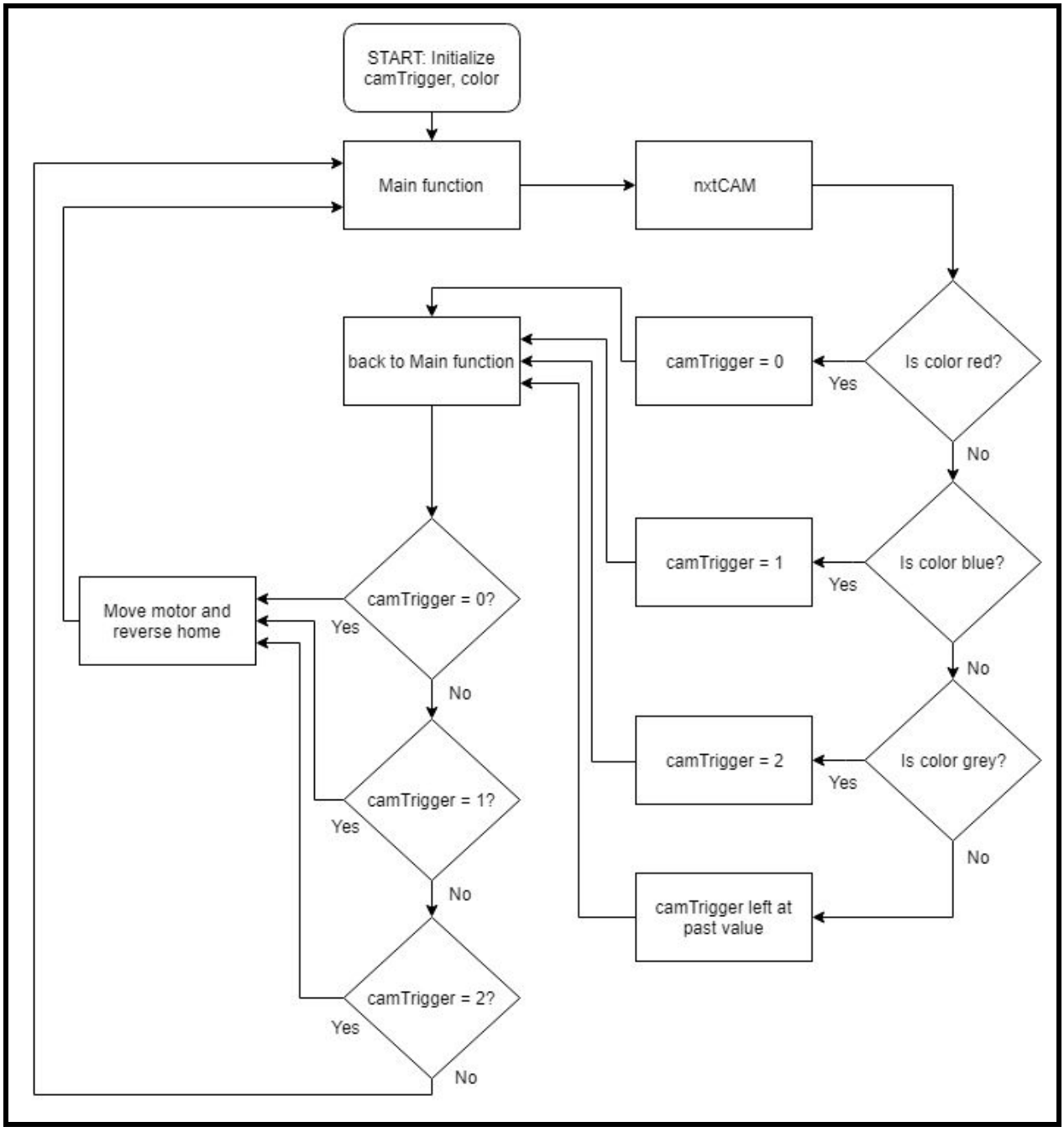


Figure 5.1 Flow-Chart of the Software Implementation for the STICK Design

The arduino in the EVShield was programmed to autonomously drop the LEGO bricks into the cups. This was done in a few steps in the main loop, discussed below.

Firstly, the nxtCAM function was called. This function would prompt the NXTcam to take a picture and look for blobs with colors that match the stored color in the camera. Depending on the color of blob found, the function would return 0, 1 or 2 for red, blue and grey colors respectively. This number is then passed onto the main function.

The main function checks for the number passed by the nxtCAM function. Depending on the number, the Motor 1 moves reverse and forward for a set degree. For red bricks, the motor moves 43 degrees. For blue bricks, the motor moves 54 degrees. For the grey bricks, the motor moves 60 degrees. All the movements are done at a speed setting of 25.

# 6. Discussion of team roles, mentorship and issues

## 6.1 Team Roles

Before constructing the sorting design, each member were given roles based on their skills learned from the laboratories. It was agreed that the work should be split up between technical work and mechanical work amongst the team members. The division of work between the team members can be seen in Table 6.1.

Table 6.1 Jobs assigned to each group member

| Team Member | Jobs |
|---|---|
| Sharfaraz | - Program and Mechanical brainstorming<br>- Construction of the claw and arm extension<br>- Camera/sensor control<br>- Primary programmer |
| Benjamin | - Mechanical brainstorming<br>- Construction of the claw and arm extension<br>- Maintenance programmer |
| Jhomar | - Program brainstorming<br>- Construction of the base<br>- Secondary/Maintenance programmer |

## 6.2 Mentorship

Original design followed the pick and drop guidelines, creating a system that has the ability to pick and place a block on a conveyor belt into a specified cup. Upon our design, we were told that the design could be made simpler by creating a system that “pushes” the blocks in to the cup. Our group was initially against this, but complications with our first design caused us to convert into creating a simple push design.

Other than being mentored by the teaching assistant, there was a mentorship amongst the group members to allow for a smooth relationship between mechanical, electrical and software implementations. Before creating the design, Jhomar and Sharfaraz would brainstorm on how the code would work and the goals the final design would have. This information was given to Benjamin and the design of the mechanical components were made. Benjamin would relay any feedback on how the mechanical components work to adjust the degree of freedom that motors will be able to rotate. Sharfaraz and Jhomar would make adjustments to the code according to the mechanical feedbacks given.

## 6.3 Issues

### 6.3.1 Mechanical Issues

Our first design, LMPY, involved 3 motor controls: base, arm extension, and claw. The issue of the first design is that the weight of the motors caused the claw to bend over the conveyor belt knocking down the cups. As a solution, more pieces were added to reinforce the system, but in the end we were unable to prevent the claw from bending over. This set back influenced the group to change the goal of the design from creating a pick and place machine to creating a sorting machine.

The second design, STCK, was created using a simple design where the system would use one motor with a long arm to push the blocks accordingly. The issue with this design is that the arm would be resting on the conveyor belt. Although the “brake hold” instruction from the EVShield causes the motor to retain its position, the mechanical arm was light enough to submit to the motion of the conveyor. Another problem we had with this design was that the bricks would also get stuck on the hook at the end of our mechanical arm. This did not occur to us until the demonstration. Another mechanical issue presented itself during the demonstration when the bricks would sometimes get stuck between the brick positioner mount of the camera. These mechanical issues could have been prevented if more time was spent on the simple design.

### 6.3.2 Technical Issues

There were many technical issues revolving around the camera sensor used in the design. It was founded while testing the camera, that the camera was unable to the size of the blocks. The camera was also unable to read the grey blocks during the calibration stage. This issue could be caused by the signal from the grey blocks being similar to the background colour of the conveyor belt. Before the demonstration, the COM connection of the camera would not open and therefore it was unable to be tested before running the program. During the demonstration for our design, the camera connection to the EVShield would cut out and had to be reconnected every time. This problem could have been caused by a faulty connection between the NXT camera and an EVShield. A solution to this problem during the demonstration was to replug the camera and reset the EVShield everytime the connection would randomly cut out.

# 7. Conclusions and Recommendations

## 7.1 Conclusion

From this project, the group was able to implement the skills that were developed throughout the course in a solution that mimicked a realistic problem that, as mechatronics engineers, could be faced in industry. During the competition portion of the project, the group was unsuccessful at delivering the required metrics. This was in part due to technical difficulties with the NXTcam during the competition. Having not been able to spend enough time calibrating and testing the robot was another reason why the team was unsuccessful. This was in part, due to the fact that various prototypes were tested before a final design was decided upon which in turn, lead to a lack of time to work on the final design and its implementation. If given additional time, the outcome of the competition likely would have differed. It was also noted that during the competition, slippage occured within the conveyor; this was a problem as it meant that the block which had already triggered the function for the specific colour/shape/size combination would move at a slower rate. Since the functions were programed such that a time delay would allow for the lego block to be knocked into their respective cups under the assumption that the blocks would be moving at a constant rate, any variation in the time was detrimental to the performance of the arm. It was also observed that the glare on the lego bricks (especially on the grey bricks), significantly skewed the results of NXTcam as rather than grey the NXTcam would read white. Lastly, due to the unreliability of the lego servo motors and their ability to brake, the arm would begin to drag from the friction of the conveyor; although rollers were added at the contact points (as can be seen in figure 2.1) of the arm and the conveyor, it was inadequate at mitigating the effects of the conveyor friction.

## 7.2 Recommendations

Several improvements could be suggested from the experience of this project. Firstly, since the EVShield repeatedly failed to consistently power the NXTcam during the competition, it is desired to have better shields to ensure smooth operation. Moreover, the calibration of the NXTcam was very inconsistent as the camera was sensitive to the lighting conditions. As such, a higher resolution camera with a better ability to mitigate different lighting conditions would make the sorting process of the lego bricks more rigorous. On the other hand, the lighting conditions of the actual competition environment could be controlled since the cameras were calibrated in the competition environment beforehand, but did not perform reliably. Furthermore, the motors provided were not accurate with the angle instructions encoded into the arduino file passed onto the EVShield. With repeated instructions to move and reverse at certain angles, the motors would slowly start to over/undershoot the mark. This only compounded with additional movements. As such, motors with precise encoders and stronger brake functionality is desired. In terms of improvements to the design, a light sensor could have been incorporated at arm. The purpose of this sensor would be to detect if a lego brick has touched the arm. Then arm would then move to the desired angle and stay there until the light sensor has detected that

the lego is no longer present on the conveyor. This would have resulted in a more robust solution.

# References

[1] V. Chan, "MEC830 Design Project Outline", *Courses.ryerson.ca*, 2018. [Online]. Available: https://courses.ryerson.ca/d2l/le/content/192502/viewContent/2115805/View. [Accessed: 01- Dec- 2018].

# Appendix: Program Listings

## Final Program Code (STCK)

```
//EVShield NXTcam
#include<Wire.h>
#include<EVShield.h>
#include<EVs_NXTCam.h>
#include<USART.h>

EVShield evshield(0x34, 0x36);
EVs_NXTCam myCam;//make object myCam

//declarevaribales
int nblobs;
int encoder;
int camTrigger;
uint8_t color[8];
uint8_t left[8];
uint8_t top[8];
uint8_t bottom[8];
uint8_t right[8];


void setup() {
  camTrigger = 99;
  initUSART();
  printString("startingNXTCam program");
  printString("\r\n");
  evshield.init(SH_HardwareI2C);
  myCam.init(&evshield, SH_BBS1);
  //if there was previous runof this program, disable it.
  myCam.disableTracking();

  //set up myCam for Object mode and sort by size.
  myCam.selectObjectMode();
  myCam.sortSize();
  myCam.enableTracking();
  delay(1000);
}

void loop() {
```

```
  nxtCAM();
  Serial.print("Motor status = ");
  encoder = evshield.bank_a.motorGetEncoderPosition(SH_Motor_1);
  Serial.println(encoder);
  printString("\r\n");
  delay(1000);

  if ( camTrigger == 0 || evshield.getButtonState(BTN_LEFT) == true ) {
    //reset trigger
    camTrigger = 99;
    //move arm
    evshield.bank_a.motorRunDegrees(SH_Motor_1, SH_Direction_Reverse, 25, 43,
SH_Completion_Wait_For, SH_Next_Action_Brake);
    Serial.print("Motor status = ");
    encoder = evshield.bank_a.motorGetEncoderPosition(SH_Motor_1);
    Serial.println(encoder);
    printString("\r\n");
    delay(20000);
    //return
    evshield.bank_a.motorRunDegrees(SH_Motor_1, SH_Direction_Forward, 25, 43,
SH_Completion_Wait_For, SH_Next_Action_Brake);
  }
  else if ( camTrigger == 1 || evshield.getButtonState(BTN_RIGHT) == true ) {
    //reset trigger
    camTrigger = 99;
    //move arm
    evshield.bank_a.motorRunDegrees(SH_Motor_1, SH_Direction_Reverse, 25, 54,
SH_Completion_Wait_For, SH_Next_Action_Brake);
    Serial.print("Motor status = ");
    encoder = evshield.bank_a.motorGetEncoderPosition(SH_Motor_1);
    Serial.println(encoder);
    printString("\r\n");
    delay(20000);
    //return
    evshield.bank_a.motorRunDegrees(SH_Motor_1, SH_Direction_Forward, 25, 54,
SH_Completion_Wait_For, SH_Next_Action_Brake);
  }
  else if ( camTrigger == 2 || evshield.getButtonState(BTN_GO) == true ) {
    //reset trigger
    camTrigger = 99;
    //move arm
    evshield.bank_a.motorRunDegrees(SH_Motor_1, SH_Direction_Reverse, 25, 60,
SH_Completion_Wait_For, SH_Next_Action_Brake);
```

```
    Serial.print("Motor status = ");
    encoder = evshield.bank_a.motorGetEncoderPosition(SH_Motor_1);
    Serial.println(encoder);
    printString("\r\n");
    delay(20000);
    //return
    evshield.bank_a.motorRunDegrees(SH_Motor_1, SH_Direction_Forward, 25, 60,
SH_Completion_Wait_For, SH_Next_Action_Brake);
  }
}

int nxtCAM() {
  myCam.issueCommand('J');//lockbuffer
  delay(500);

  myCam.getBlobs(&nblobs, color, left, top, right, bottom);
  delay(500);

  myCam.issueCommand('K');//unlockbuffer
  printString("numberofblobs:");
  printWord(nblobs);
  printString("\r\n");
  uint8_t i;
  for (int i = 0; i < nblobs; i++) {
    if (color[i] == 0) {
      printString("redblobfoundat: X: ");
      printWord((left[i] + right[i]) / 2);
      printString("Y: ");
      printWord((top[i] + bottom[i]) / 2);
      printString("\r\n");
      camTrigger = 1;
    }
    if (color[i] == 1) {
      printString("blue blob found at: X: ");
      printWord((left[i] + right[i]) / 2);
      printString("Y: ");
      printWord((top[i] + bottom[i]) / 2);
      printString("\r\n");
      camTrigger = 2;
    }
    if (color[i] == 2) {
      printString("grey blob found at: X: ");
      printWord((left[i] + right[i]) / 2);
```

```
      printString("Y: ");
      printWord((top[i] + bottom[i]) / 2);
      printString("\r\n");
      camTrigger = 3;
    }
  }
  return camTrigger;
}
```

## Program Code (LMPY)

```
//EVShield NXTcam
#include<Wire.h>
#include<EVShield.h>
#include<EVs_NXTCam.h>
#include<USART.h>

EVShield evshield(0x34, 0x36);
EVs_NXTCam myCam;//make object myCam

//declarevaribales
int nblobs;
int camTrigger;
uint8_t color[8];
uint8_t left[8];
uint8_t top[8];
uint8_t bottom[8];
uint8_t right[8];


void setup() {
  //NXTCam port = BANK B BBS1
  //Base motor = BANK B M2
  //Link motor = BANK B M2
  //Gripper motor = BANK A M1
  initUSART();
  printString("startingNXTCam program");
  printString("\r\n");
  evshield.init(SH_HardwareI2C);
  myCam.init(&evshield, SH_BBS1);
  //if there was previous runof this program, disable it.
  myCam.disableTracking();
```

```
  //set up myCam for Object mode and sort by size.
  myCam.selectObjectMode();
  myCam.sortSize();
  myCam.enableTracking();
  delay(1000);
}

void loop() {
  nxtCAM();
  delay(5);
  if ( camTrigger == 1) {
    //reset trigger
    camTrigger = 0;
    //drive gripper
    driveMotor(1, 1, 1);
    delay(350);
    stopMotor(1, 1);
    delay(500);
    //drive base motor
    driveMotor(2, 2, 1);
    delay(2000);
    stopMotor(2, 2);
    delay(500);
    //drive link motor
    driveMotor(1, 2, 1);
    delay(500);
    stopMotor(1, 2);
    delay(500);

    //Return to initial position
    //drive gripper
    driveMotor(1, 1, 2);
    delay(350);
    stopMotor(1, 1);
    delay(500);
    //drive base motor
    driveMotor(2, 2, 2);
    delay(2000);
    stopMotor(2, 2);
    delay(500);
    //drive link motor
    driveMotor(1, 2, 2);
```

```
  delay(100);
  stopMotor(1, 2);
  delay(500);
}

else if ( camTrigger == 2) {
  //reset trigger
  camTrigger = 0;
  //drive gripper
  driveMotor(1, 1, 1);
  delay(350);
  stopMotor(1, 1);
  delay(500);
  //drive base motor
  driveMotor(2, 2, 1);
  delay(2000);
  stopMotor(2, 2);
  delay(500);
  //drive link motor
  driveMotor(1, 2, 1);
  delay(500);
  stopMotor(1, 2);
  delay(500);

  //Return to initial position
  //drive gripper
  driveMotor(1, 1, 2);
  delay(350);
  stopMotor(1, 1);
  delay(500);
  //drive base motor
  driveMotor(2, 2, 2);
  delay(2000);
  stopMotor(2, 2);
  delay(500);
  //drive link motor
  driveMotor(1, 2, 2);
  delay(100);
  stopMotor(1, 2);
  delay(500);
}

else if ( camTrigger == 3) {
```

```
    //reset trigger
    camTrigger = 0;
    //drive gripper
    driveMotor(1, 1, 1);
    delay(350);
    stopMotor(1, 1);
    delay(500);
    //drive base motor
    driveMotor(2, 2, 1);
    delay(2000);
    stopMotor(2, 2);
    delay(500);
    //drive link motor
    driveMotor(1, 2, 1);
    delay(500);
    stopMotor(1, 2);
    delay(500);

    //Return to initial position
    //drive gripper
    driveMotor(1, 1, 2);
    delay(350);
    stopMotor(1, 1);
    delay(500);
    //drive link motor
    driveMotor(1, 2, 2);
    delay(200);
    stopMotor(1, 2);
    delay(500);
    //drive base motor
    driveMotor(2, 2, 2);
    delay(2100);
    stopMotor(2, 2);
    delay(500);
  }

}

int nxtCAM() {
  myCam.issueCommand('J');//lockbuffer
  delay(500);

  myCam.getBlobs(&nblobs, color, left, top, right, bottom);
```

```
  delay(500);

  myCam.issueCommand('K');//unlockbuffer
  printString("numberofblobs:");
  printWord(nblobs);
  printString("\r\n");
  uint8_t i;
  for (int i = 0; i < nblobs; i++) {
    if (color[i] == 0) {
      printString("redblobfoundat: X: ");
      printWord((left[i] + right[i]) / 2);
      printString("Y: ");
      printWord((top[i] + bottom[i]) / 2);
      printString("\r\n");
      camTrigger = 1;
    }
    if (color[i] == 1) {
      printString("blue blob found at: X: ");
      printWord((left[i] + right[i]) / 2);
      printString("Y: ");
      printWord((top[i] + bottom[i]) / 2);
      printString("\r\n");
      camTrigger = 2;
    }
    if (color[i] == 2) {
      printString("grey blob found at: X: ");
      printWord((left[i] + right[i]) / 2);
      printString("Y: ");
      printWord((top[i] + bottom[i]) / 2);
      printString("\r\n");
      camTrigger = 3;
    }
  }
  return camTrigger;
}

void driveMotor(int bank, int port, int dir) {
  if ( bank == 1 ) {
    if ( port == 1) {
      if ( dir == 1) {
        evshield.bank_a.motorRunUnlimited(SH_Motor_1, SH_Direction_Forward,
SH_Speed_Full);
      }
```

```
      else {
        evshield.bank_a.motorRunUnlimited(SH_Motor_1, SH_Direction_Reverse,
SH_Speed_Full);
      }
    }
    if ( port == 2) {
      if ( dir == 1) {
        evshield.bank_a.motorRunUnlimited(SH_Motor_2, SH_Direction_Forward,
SH_Speed_Full);
      }
      else {
        evshield.bank_a.motorRunUnlimited(SH_Motor_2, SH_Direction_Reverse,
SH_Speed_Full);
      }
    }
  }

  if ( bank == 2 ) {
    if ( port == 1) {
      if ( dir == 1) {
        evshield.bank_b.motorRunUnlimited(SH_Motor_1, SH_Direction_Forward,
SH_Speed_Full);
      }
      else {
        evshield.bank_b.motorRunUnlimited(SH_Motor_1, SH_Direction_Reverse,
SH_Speed_Full);
      }
    }
    if ( port == 2) {
      if ( dir == 1) {
        evshield.bank_b.motorRunUnlimited(SH_Motor_2, SH_Direction_Forward,
SH_Speed_Full);
      }
      else {
        evshield.bank_b.motorRunUnlimited(SH_Motor_2, SH_Direction_Reverse,
SH_Speed_Full);
      }
    }
  }
}

void stopMotor(int bank, int port) {
  if ( bank == 1 ) {
```