Advanced

Mechatronics Project

Report

Propeller Project Proposal: Advanced Mobile Robotic Object Sorter

Sotomi Oluwadamilola - ots2014

El Tannir, Omar - oe2104

Ravi, Yukteshwar - yr2364

Ragupathy,Sai Nishit - sr6931

TABLE OF CONTENT

1.0. ABSTRACT

This report details the advancement of the Advanced Mobile Robotic Object Sorter, utilizing the Parallax Propeller's multicore system and Pixy2 camera for improved color-based sorting. The integration of vision-based detection, PID line-following for navigation, and a precise gripper mechanism significantly enhances sorting accuracy and efficiency. This project aims to streamline automated sorting processes by leveraging real-time colour tracking and multicore processing for optimized operation.

2.0. INTRODUCTION

Automated sorting systems are pivotal in various sectors, necessitating innovations for enhanced efficiency and accuracy. Our project introduces an Advanced Mobile Robotic Object Sorter, incorporating the Parallax Propeller and Pixy2 camera to elevate sorting capabilities through advanced color detection and precise object handling. This report outlines the project's phased approach, focusing on enhancing detection and sorting mechanisms, aiming to improve operational efficiency in automated sorting applications.

3.0. PARTS/ALGORITHM AND EXPLANATION

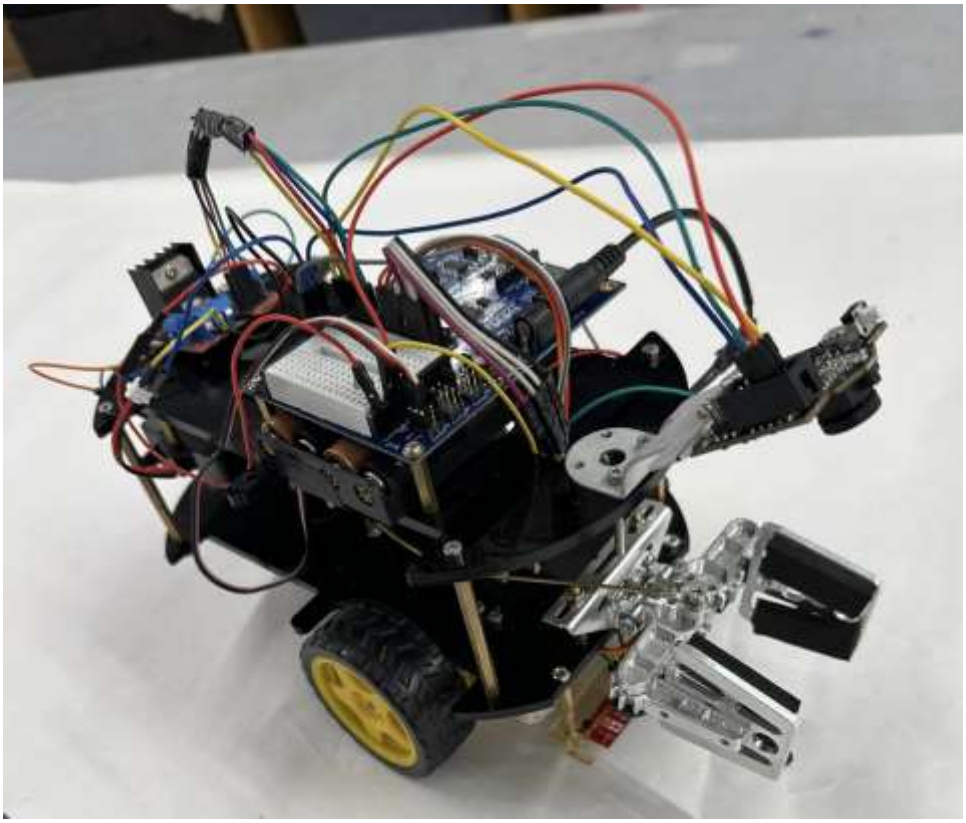


Fig 3.1. **Advanced Mobile Robotic Object Sorter**

3.1 Parallax Propeller Microcontroller:

Fig.3.2 **Parallax Propeller Microcontroller**

The Parallax Propeller is a robust multicore microcontroller, distinguished by its architecture that comprises 8 independent processing cores, or cogs, enabling parallel execution of tasks. This feature is pivotal for high-speed, simultaneous processing, optimizing the microcontroller's performance across a multitude of robotic functionalities including motor control, sensor input processing, and complex communication protocols. The architecture notably enhances simultaneous input/output operations across multiple pins, which is crucial for intricate interactions with an array of sensors and actuators. Designed to be user-friendly, the Propeller's programming environment caters to developers at varying levels of expertise, fostering an inclusive environment for robotics innovation.

3.2. Pixy2 Camera System:

Fig 3.3. **Pixy2 Camera System**

The Pixy2 camera stands out as a compact, fast vision sensor endowed with four distinct modes of operation, enhancing its utility across various robotic applications. Among these, the Color Connected Components mode is primarily leveraged in this project, where it excels in detecting and tracking objects based on their colors. Additionally, the Line Tracking mode facilitates advanced line detection capabilities, the Video mode converts Pixy2 into a standard video camera, and the Pan Tilt mode, where the camera is mounted on a mechanism, allows for dynamic movement or rotation along two axes. For our application, the Color Connected Components mode is pivotal, enabling real-time, accurate object detection and tracking based on color differentiation. This capability is integral for tasks requiring precise color-based sorting or navigation, providing a seamless interface for setting color signatures and facilitating direct connection with a range of microcontrollers, including the Parallax Propeller, for streamlined integration into robotics projects.

3.3. PID Line-Following Algorithm:

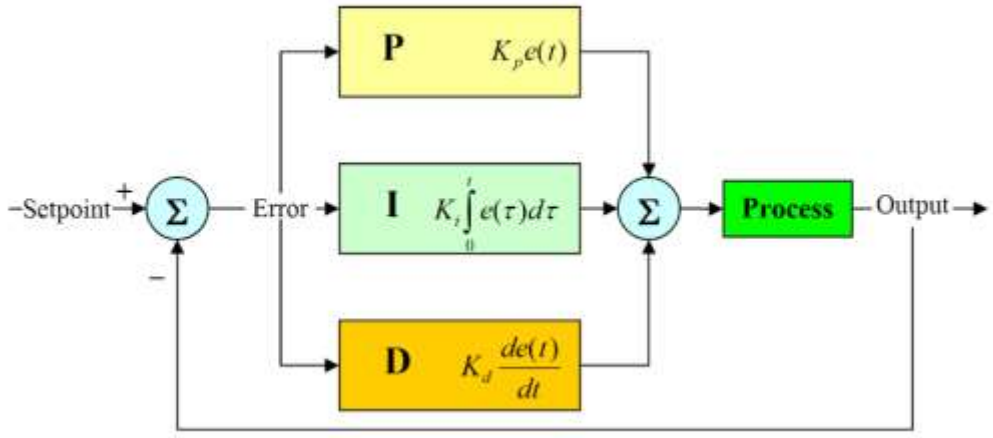


Fig 3.4. PID Line-Following Algorithm

PID (Proportional, Integral, Derivative) control is a feedback mechanism used to ensure precise and stable navigation. In the context of this project, the PID algorithm helps the robot follow a line with high accuracy, adjusting its path based on continuous sensor feedback to correct deviations, ensuring smooth and reliable movement along predefined paths. The algorithm's parameters can be finely tuned to match the specific dynamics of the robotic platform, ensuring optimal performance. It is integral in minimizing errors and enhancing the sorter's ability to navigate complex routes without human intervention.

3.4 DC Motors

Fig 3.5 DC Motors

DC motors translate direct electrical energy into mechanical motion and are instrumental across a plethora of applications, from miniature models to substantial industrial equipment. Their operation is based on electromagnetic principles, with a design that offers high torque at low speeds, essential for robotics and automated machinery.

3.5 QTR 8RC Sensor:

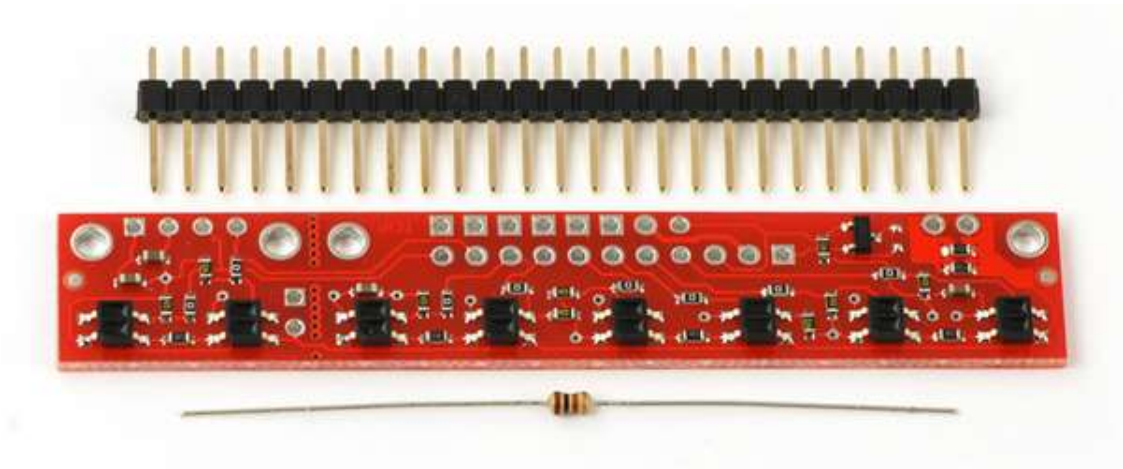


Fig.3.6. QTR 8RC Sensor

This sensor array comprises eight infrared reflectance sensors, optimized for line-following and edge-detection tasks within robotic frameworks. It determines line presence or edges by measuring infrared light reflection, providing essential feedback for precise navigation.

3.6 L298N **H-Bridge Motor Driver:**

Fig 3.7. L298N **H-Bridge Motor Driver**

The L298N Dual H-Bridge Motor Driver module facilitates the independent operation of two DC motors, supporting both forward and backward movements. Its H-Bridge configuration is key for reversing motor supply voltage polarity, crucial for powering diverse automated and robotic applications.

3.7. Li-ion Battery:

Fig 3.8. Li-ion Battery

Characterized by their high energy density and longevity, Lithium-ion batteries are the project's power source, facilitating the movement of lithium ions between electrodes through an electrolyte. Their widespread use in portable electronics, electric vehicles, and energy storage solutions speaks to their efficiency and robustness.

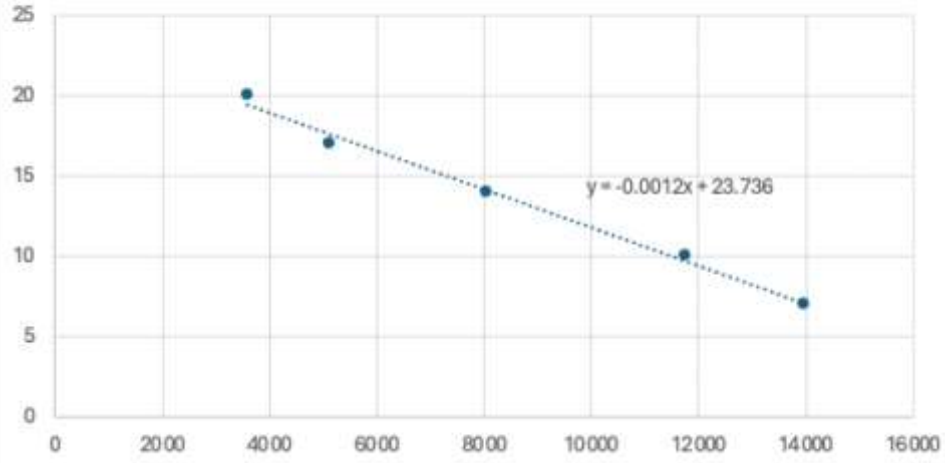
3.8. **Gripper Mechanism:**

Fig 3.10. Gripper Mechanism

The gripper mechanism is a robotic end-effector designed to grasp and manipulate objects. In this project, it's synchronized with the vision system (Pixy2) to pick up, hold, and sort objects based on color. The mechanism is versatile, able to handle objects of various sizes and shapes, making it an essential component for automated sorting tasks. It is engineered for durability and precision, ensuring reliable operation over extended periods. Additionally, the gripper's design allows for adjustments to accommodate different object geometries, enhancing the sorter's versatility.

4.0 DISTANCE ESTIMATER

Area	d	W	h	
3600	20	60	60	19.416
5120	17	64	80	17.592
8064	14	72	112	14.0592
11760	10	80	147	9.624
13984	7	92	152	6.9552



In the Advanced Mobile Robotic Object Sorter project, a novel approach is implemented for estimating the distance to objects using the Pixy2 camera, a cornerstone of the sorter's detection system. The Pixy2 camera, when it identifies an object, automatically outlines it with a bounding box, the dimensions of which—width and height—are dynamically calculated via a function available in the Pixy2's library. These dimensions are pivotal; as the object moves closer or further from the camera, the size of the bounding box changes accordingly. This variation is critical for our distance estimation strategy, where we employ a linear function that takes these measurements into account. The function correlates the changing dimensions of the detected object with its distance from the camera, enabling the system to determine how far an object is with remarkable accuracy.

This mechanism is integral to the sorter's operation, particularly in automating the activation of the gripper mechanism. As an object approaches and its bounding box dimensions increase, signifying that it is getting closer, the linear function predicts the object's proximity based on the changes in size. Once the object is within a predefined threshold distance—indicative of it being sufficiently close—the system triggers the gripper mechanism. This activation is not arbitrary but is based on precise distance estimations calculated through the linear function, ensuring the gripper engages objects at the optimal moment for efficient sorting. This distance estimator is a testament to the project's innovative use of technology to enhance the accuracy and reliability of automated sorting, demonstrating a sophisticated integration of computer vision and mechanical actuation in a real-world application.

5.0. SELF-ALIGNMENT PROCESS

The self-alignment system leverages the Pixy2 camera's object detection capabilities to dynamically adjust the robot's orientation and position in relation to a targeted object. The system ensures that the robot aligns itself correctly with the object before initiating a forward movement to grip or interact with the object. This process involves real-time analysis of visual data, adaptive movement control, and precise threshold management to achieve optimal alignment.

Operational Details:

Object Detection and Distance Measurement: The Pixy2 camera continuously scans its field of view for objects that match predefined characteristics, such as color signatures. When an object is detected, the system calculates its distance from the camera using size metrics (e.g., width and height in pixels), comparing it to a set distance threshold encoded in the software. This distance threshold ensures that the robot only attempts to align with and approach objects that are within a reachable range.

Deviation Analysis and Threshold Setting: Alongside distance, the system calculates the deviation of the detected object from the center of the camera's field of view. This deviation is assessed against two critical thresholds: a left threshold and a right threshold. These thresholds are determined through empirical testing to define the acceptable alignment window. Specifically, they represent the maximum allowable deviation to the left and right when the object is considered to be directly in front of the robot's gripper.

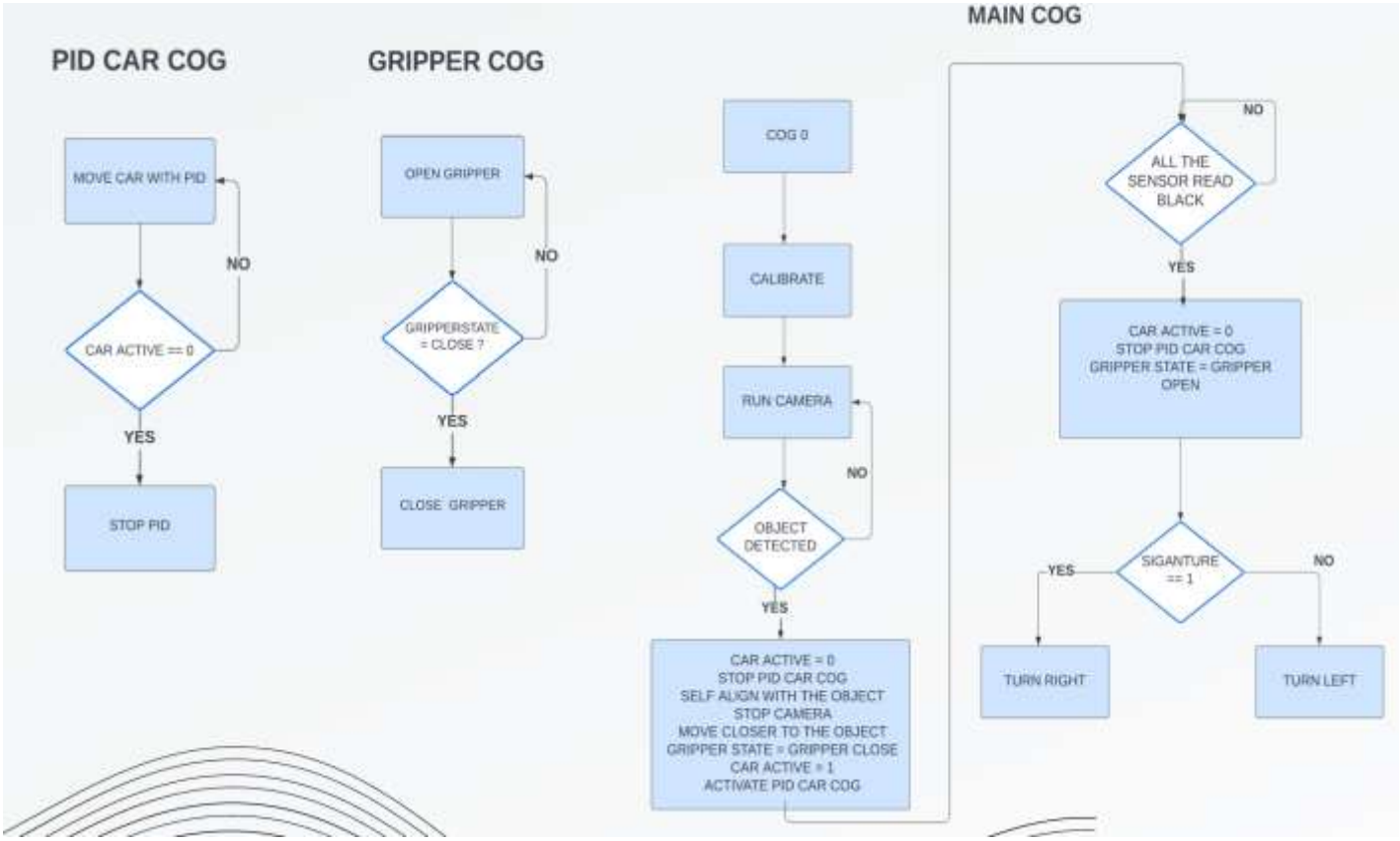
Adaptive Alignment Control:

If the object's deviation falls outside the acceptable alignment window, the robot engages in corrective movement. Depending on whether the deviation indicates that the object is too far to the left or right, the robot will adjust its orientation by moving laterally in the appropriate direction. This realignment process is iterative, with the robot continuously re-evaluating the object's position and making adjustments until the deviation falls within the specified thresholds.

Forward Movement and Object Interaction:

Once the object is deemed to be correctly aligned within the gripper's operational zone (i.e., the deviation is within the acceptable thresholds), the robot proceeds to move forward towards the object. Upon reaching the appropriate distance, as dictated by the initial distance threshold, the robot activates its gripping mechanism to securely interact with the object, completing the self-alignment and object retrieval process.

6.0. CODE FLOW & EXPLANATION



Initial Setup: The code begins by initializing communication with the Pixy2 camera and setting up essential parameters, including pins for communication and parameters for object detection. If the Pixy2 camera starts successfully, the robot proceeds with calibration and prepares its various components for operation.

Parallel Operations through Cogs: To efficiently manage its tasks, the robot leverages the Propeller microcontroller's ability to run multiple processes in parallel, referred to as "cogs".

Specifically, it starts driving along a predetermined path using PID control in one cog (driveCarCog).Begins gripper control in another cog to manage the opening and closing of the gripper based on the task at hand. Executes the camera and object detection logic in the main thread (or "cog") to continually monitor for recognizable objects.

Object Detection and Handling: As the robot moves, the camera function (camera) is continuously running, scanning for pre-trained objects. When an object is detected within a specific distance, the robot halts its movement (stopMovement) and prepares to interact with the object. Depending on the object's signature, which is determined by the camera, the robot may adjust its alignment to ensure precise interaction with the object.

Once properly aligned and the object is within grasp, the robot uses its gripper to pick up the object (pickItem).

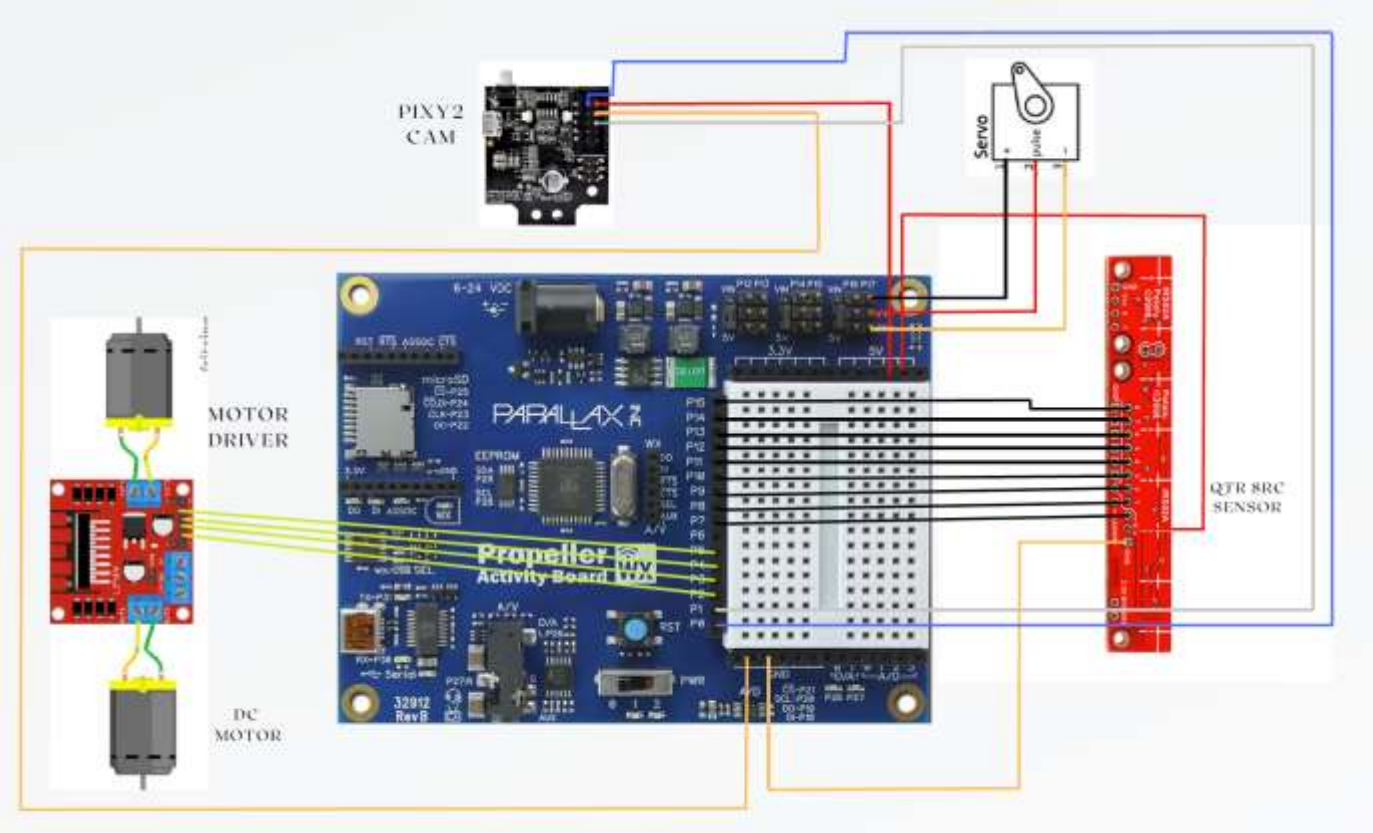
Navigational Adjustments: After picking up the object, the robot resumes its journey. Depending on the detected object's characteristics (in this case, its signature), the robot makes a decision on where to place the object:

If the object's signature corresponds to a specific category, the robot may turn left or right to place the object in the designated location (turnLeft90Degrees or turnRight90Degrees).

Conclusion of Operation: Once the object is placed in the correct location, the robot opens the gripper to release the object and potentially concluding its task, marking the end of the program's current cycle.

This structure allows the robot to perform complex navigation and object manipulation tasks in real-time, by dividing the work among different processes that run concurrently, enhancing efficiency and responsiveness

7.0. CIRCUIT SCHEMATIC



In the Advanced Mobile Robotic Object Sorter project, the Parallax Propeller microcontroller stands at the heart of the system, orchestrating operations by connecting to the Pixy2 camera via UART for advanced object and color detection. The QTR 8RC sensors, interfaced directly with the microcontroller, enable precise line tracking, while DC motors, controlled by the L298N

motor driver, under the microcontroller's command, facilitates movement and direction. A servo motor, also governed by the microcontroller, operates the gripper mechanism for object sorting. This setup is powered by a Li-ion battery, ensuring a reliable energy source for the system's

operations. The interconnections between these components form a complex yet efficient network, enabling the sorter to execute its functions with high precision and efficiency, showcasing a sophisticated integration of electronic and mechanical components in robotic sorting technology.

8.0 APPENDIX

Main Code:

```
#include "simpletools.h"
#include "millis.h"
#include "qtr_sensors.h"
#include "PID.h"
#include "pixy2.h"
#include "servo.h"

#define RX2 1                       //rx pin for the pixy 2 camera
#define TX2 0                       //Tx pin for the pixy 2 camera
#define SIG_MAP CCC_SIG1 | CCC_SIG2 | CCC_SIG3
#define CC_MAP CCC_COLOR_CODES | SIG_MAP
#define LF_SPEED 100.0
#define MIN_DISTANCE_1 12           //Min Distance from camera to object 1 before motors stops
#define MIN_DISTANCE_2 12           //Min Distance from camera to object 2 before motors stops

#define GRIPPER_SERVO_PIN 17        //defines servo pin
#define GRIPPER_OPEN 1800           //defines servo open position
#define GRIPPER_CLOSE 0             //defines servo open position
```

```
volatile int gripperState = GRIPPER_OPEN;     //global variable to store gripper state
volatile int carActive = 1;                   //global variable to store car active state
volatile int cameraActive = 1;                //global variable to store camera active
volatile int checkAlignment = 0;              //global variable to store checkalignment state
volatile int finalSignature = 0;              //global variable to store the signature of the object picked


int alignmentCounter = 0;                     //global variable to store the alignment count
int driveCarCog;                              //global variable to store the cog value running drivecar function

uint16_t frameWidth = 316;                    // width of Pixy frame
uint16_t frameHeight;                         // height of Pixy frame
block_t b;                                    // target (block) values

void showError(int32_t code);
void driveCar();                              //function that moves the car
void camera();                                //function that runs the camera
void showSignatures();                        //function that constantly checks the signature of the object in view
void stopEverything();                        //stops motor and pwm activity
void controlGripper();                        //function to handle gripper
void pickItem();                              //function to control gripper state
void turnLeft90Degrees();                     //function to turn the car left
void turnRight90Degrees();                    //function to turn the car right

int main() {

   uint32_t ec = pixy2_start(RX2, TX2, 19200);
   if (ec != RESULT_OK) {
      showError(ec);
```

```
    while (1) {}
} else {
    print("Pixy2 Serial Test\n\n");
}

pause(1000);
calibrateSensors();                     // Perform calibration on sensors

initMillis();                           //initiate millis
driveCarCog=cog_run(driveCar, 256);          // starts pid and drives car on a separate cog
cog_run(controlGripper, 128);               // starts gripper control on a separate cog
camera();                               // starts the camera function on main cog

pause(1000);
motor_drive(200,200); //150 before
pause(200);
motor_stop();

pickItem();
pause(1000);
carActive = 1;                          //resumes car movement after picking item
pause(1000);
driveCarCog=cog_run(driveCar, 256);          // starts pid and drives car again on a separate cog

pause(1000);


int consecutiveCounts = 0;              // Initialize the counter to track consecutive readings above threshold
 while(1) {
    int allAboveThreshold = 1;          // Flag to check if all sensors are above threshold
```

```
 for(int i = 0; i < sensorCount; i++) {
    if(sensorValues[i] <= 850) {
       allAboveThreshold = 0;            // If any sensor is not above threshold, set flag to 0
       break;                          // Exit the loop early if a sensor doesn't meet the condition
    }
 }
 if(allAboveThreshold) {
    consecutiveCounts++;                // Increments the counter if all sensors are above the threshold
    if(consecutiveCounts >= 2) {
       break;                          // Exit the while loop if the condition has been true for 2 iterations
    }
 } else {
    consecutiveCounts = 0;              // Reset the counter if the condition is not met
 }
 pause(50);                             // Small pause to prevent hogging the CPU
}

 carActive = 0;                         //stops car movement
 motor_stop();
 cog_end(driveCarCog);                  //stops the car cog

 pause(2000);                           //makes a decision based on object signature
 if(finalSignature==1){
  turnRight90Degrees();
  }
 else if(finalSignature==2){
  turnLeft90Degrees();
  }
 pause(1000);
 gripperState = GRIPPER_OPEN;           //opens gripper to release objects
```

```
    return 0;
}



/*
function to drive car using PID control. Depends on the state of carActive variable
*/
void driveCar() {
    while (carActive) {
        pause(80);
        double linePosition = readLine();           // Calculate the line position
        double linePosition_New = ((linePosition) * (200.0) / 7000.0) - 100.0;
        double output = ComputePID(linePosition_New);

        int lsp = LF_SPEED - output;
        int rsp = LF_SPEED + output;
        motor_drive((int)lsp, (int)rsp);
    }
}

/*
function to control the camera. Depends on the state of cameraActive variable
*/
void camera() {
    while (cameraActive) {
        showSignatures();
    }
}
```

```
void showSignatures() {
  int32_t nsigs;
  float distance1, distance2, distance3;
  const int leftThreshold = -22;                    //value gotten based on test
  const int rightThreshold = 14;
  int aligned = 0;
  nsigs = pixy2_getBlocks(SIG_MAP, MAX_BLOCKS);

    if (nsigs < RESULT_OK) {
       showError(nsigs);
    } else {
       if (nsigs > 0) {
          for (uint8_t i = 0; i < nsigs; i++) {
             pixy2_extractBlock(i, &b.signature);
             int area = b.width * b.height;
             distance1 = -0.0012 * area + 23.736;
             distance2 = -0.0012 * area + 23.736;
             distance3 = -0.001 * area + 21.233;
             int deviation = b.xpos-(frameWidth/2);
            if ((b.signature == 1 && distance1 < MIN_DISTANCE_1) ||(b.signature == 2 && distance2 <
MIN_DISTANCE_2)) {
               finalSignature= b.signature;
               stopMovement();
             }
             if (checkAlignment) {
                if (deviation < leftThreshold) {
                    motor_drive(-80,80);
                } else if (deviation > rightThreshold) {
                    motor_drive(80,-80);
```

```
            } else {                    //if aligned, loop runs for a while to ensure accurate alignment
              if(alignmentCounter>=200){
               cameraActive = 0;            // stop the camera if aligned
               checkAlignment = 0;           // Reset alignment checking
               pause(1000);
               return;
              }
              alignmentCounter+=1;
          }
        }
      }
   }
}
}
void stopMovement() {
   carActive = 0;                       // Stops the driveCar function from running
   motor_stop();                        // Stops motors immediately
   cog_end(driveCarCog);                      // Stops the driveCar cog
   high(27);
   checkAlignment = 1;
}
void showError(int32_t code) {
   print("Pixy error: %d\n", code);
}
/*
function to control the gripper
*/
void controlGripper() {
   while(1) {
      if(gripperState == GRIPPER_CLOSE) {
```

```
            servo_angle(GRIPPER_SERVO_PIN, GRIPPER_CLOSE);
        } else if(gripperState == GRIPPER_OPEN) {
            servo_angle(GRIPPER_SERVO_PIN, GRIPPER_OPEN);
        }
    }
}

/*
function to control the gripperstate
*/
void pickItem(){
  gripperState = GRIPPER_CLOSE;
  }
void turnRight90Degrees(){
  motor_drive(200, 0);
  pause(350);
  motor_stop();
  }
void turnLeft90Degrees(){
  motor_drive(0, 200);
  pause(350);
  motor_stop();
  }
```