MECH 472 - MECHATRONICS AND AUTOMATION

TEAM VISION X

Project Report

Submitted by:

Patrick Bou Rjeili [27730233]

Ahmed Hashem [40067404]

Nazih Mallouk [40119369]

Benjamin Martel [40034821]

Vinura Paranagama [40027111]

Department of Mechanical, Industrial and Aerospace Engineering

Gina Cody School of Engineering and Computer Science

Concordia University

Winter 2020

INTRODUCTION

The objective of this project was to build a robot to shoot a laser at an opponent robot while navigating an obstacle course while taking fire from the opposing robot. The guidance will be provided via the overhead webcam.

MAIN PROGRAM

The guidance for the system uses the RGB values (using IrfanView) of the colored circles to determine their centroids. The values are as follows:

Red = R> 200, G<100, B<100.
Green = R<80, G>200, B<150.
Blue = R<60, G>150, B>200.
Orange = R>200, 180<G<200,110<B<130.

```
int centroid_green(image &rgb, int &icg, int &jcg, int &b){
    ibyte *pg;
    int i, j, height, width;
    int mg, mig, mjg;
    int B, G, R;


    pg = rgb.pdata;

    width = rgb.width;
    height = rgb.height;

    mg = 0, mig = 0, mjg = 0;

    for (j = 0; j < height; j++){
        for (i = 0; i < width; i++){
            B = *pg;
            G = *(pg + 1);
            R = *(pg + 2);
            if (R < 80 && G > 170 && B < 150){
                mg += G;
                mig += G*i;
                mjg += G*j;
            }
            pg += 3;
        }//end of i loop
    }//end of j loop
    if (mg == 0){
        //cout << "\nCOLOR GREEN NOT FOUND";
        b = 1;
        return 1;
    }
    icg = mig / mg;
    jcg = mjg / mg;
    b = 0;
    return 0;
```

Figure 1 - Sample Centroid Calculations

The program copies the original simulation image and then converts the copied image to a grayscale image. Using the cover_objects function it removes the two robots from the image and after a threshold value of 100 and inverting, it labels the binary image with the obstacles that remained on the plane.

The robot avoids obstacles using the angle between the two colored circles on the robot and the angle between the object and the green circle (the front of the robot). The relation between angles determines whether the robot should turn eight or left.

```
for (i = 1; i < nlabels + 1; i++){
    dis_front[i] = (i_ob[i] - ig)*(i_ob[i] - ig) + (j_ob[i] - jg)*(j_ob[i] - jg);
    dis_back[i] = (i_ob[i] - ir)*(i_ob[i] - ir) + (j_ob[i] - jr)*(j_ob[i] - jr);

    ob_angle[i] = atan2((j_ob[i] - jg), (i_ob[i] - ig)) * 180 * pi_inv;
    if (ob_angle[i] < 0)ob_angle[i] = 360 - fabs(ob_angle[i]);
    //  cout << "\tdistance " << i << " = " << dis_front[i] << "\tangle " << i << " = " << ob_angle[i];

    if (dis_front[i] <= 9000) {

        avoid_obstacles(dis_front[i], dis_back[i], ob_angle[i], beta, pw_l, pw_r);
        break;
    }
    else    avoid_robot(beta, beta_op, pw_l, pw_r, icg, jcg);
```

Figure 2 - Relations Between the Centroids

The guidance program to point the robot during offensive mode works in a similar way. The program calculates the angle between the two centroids of the player robot and the angle between the two front centroids of the both robots to determine its orientation in relation to the orientation of the robot. The motor settings are adjusted accordingly.

```
if (angle1 / 10 == angle2 / 10) {
    PW_L = 1000;
    PW_R = 2000;//move forward
    if (target <= 40000) {
        l = 1;//so that it will be -N->close
        return 0;
    }
}

else if (angle1 < angle2) {// move to the left
    PW_L = 1900;
    PW_R = 1900;
}

else if (angle1 > angle2){ // move to the right
    PW_L = 1100;
    PW_R = 1100;
}

//this is needed because it will assume that angle2>angle1 and move to-N-> the left instead of right
if (angle1 >= 0 && angle1 <= 90 && angle2 >= 270 && angle2 <= 359){
    PW_L = 1100;
    PW_R = 1100;

}

//this is needed because it will assume that angle2>angle1 and move to the right instead of left
if (angle2 >= 0 && angle2 <= 90 && angle1 >= 270 && angle1 <= 359){
    PW_L = 1900;
    PW_R = 1900;

}

return 0;
```

Figure 3 - Point Robot Function

(Angle1 = angle of robot, Angle 2 = angle of line connecting 2 front centroids)

The avoid robot function works during the defensive part of the program. This program works much like the aforementioned functions. The difference between them can tell us the direction that the opponent has moved. The relative distance between the robot and the opponent is used. The robot will attempt to avoid the opponent by turning in the opposite direction to that of the robot.

```
int avoid_robot(double beta, double beta_op, int &PW_L, int &PW_R,  int &icg, int &jcg)
{

    //double iro, jro;
    int angle1, angle2, a2, d2, target;

    angle1 = (int)beta; // robot angle

    angle2 = (int)beta_op; // enemy angle
    d2 = a2 - angle2;
    if (d2>0) // enemy turned left
    {
        PW_L = 1100;
        PW_R = 1100;
    }
    else if (d2 < 0) // enemy turned right
    {
        PW_L = 1900;
        PW_R = 1900;
    }
    else if (abs(beta - beta_op) > 75 && abs(beta - beta_op) < 105)
    {
        PW_L = 1000;
        PW_R = 2000;
    }

    a2 = (int)beta_op; // remember opponent angle to measure d-angle
    return 0;
}//end of function
```

Figure 4 - Avoid Robot Function

The cover_object function needed for the centroid calculations of the obstacles.

```
int cover_object(image &grey, int is, int js)
{
    ibyte *pa;
    double r, rmax, dr, s, smax, ds, theta;
    int i, j;

    pa = grey.pdata;

    rmax = 60.0; // maximum radius of that will turn white (pixels)
    dr = 0.5; // delta radius  (pixels)
    ds = 0.5; // delta arc-length (pixels)

    // covering pixels in an outward concentic ring pattern
    for (r = 1.0; r <= rmax; r += dr) {
        smax = 2 * 3.1416*r; // maximum arc length
        for (s = 0; s <= smax; s += ds) {
            theta = s / r; // s = r*theta
            i = (int)(is + r*cos(theta));
            j = (int)(js + r*sin(theta));
            *(pa + j*grey.width + i) = 255;

            // limit (i,j) from going off the image
            if (i < 0) i = 0;
            if (i > grey.width - 1) i = grey.width - 1;
            if (j < 0) j = 0;
            if (j > grey.height - 1) j = grey.height - 1;
        }
    }

    return 0;
```

Figure 5 - Cover Object Function

PHYSICAL ROBOT

The physical robot features a fairly simple configuration, with the frame being a ⅛” thick sheet of baltic birch cut to the specified dimensions (200 x 400mm). Two plastic boxes covering the electronics are bolted to the top of the frame. Furthermore, most parts were custom designed and 3D printed, including the servo wheels, the servo mounts and the ping pong ball caster.

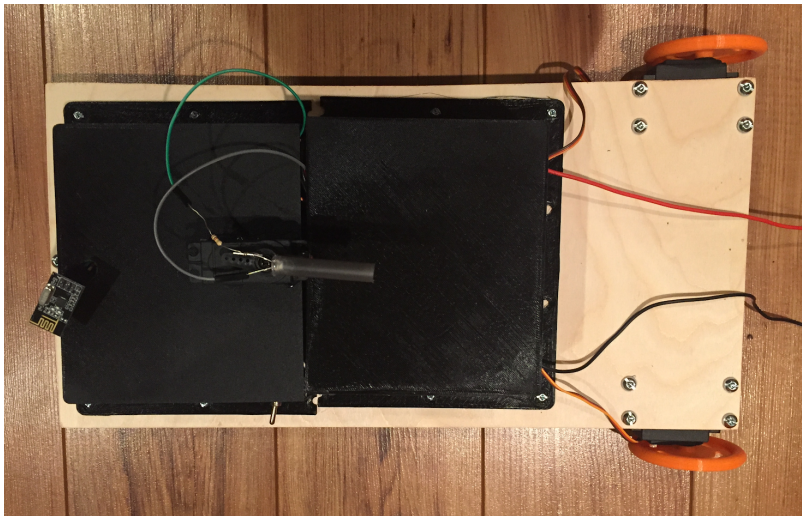

Figure 6 - Top View of Robot

Figure 7 - Bottom View of Robot

The custom designed servo mounts lock the wheel servos to the bottom surface of the robot frame. They are secured in place with nuts and bolts that pass through the 4 holes integrated into the design. Furthermore, a slot was cut into one of the side walls in order to pass the wires coming out of the bottom of the servos. Chamfers were incorporated to some surfaces for an added visual element to the design.

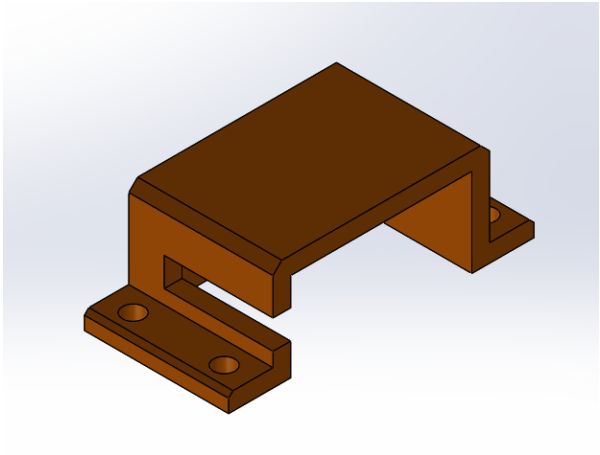

Figure 8 - Servo Mount

The servo wheels are simple wheels taken from a CAD library. The hub was modified as to match the spline on the servo, locking the wheel in place with the servo screw.

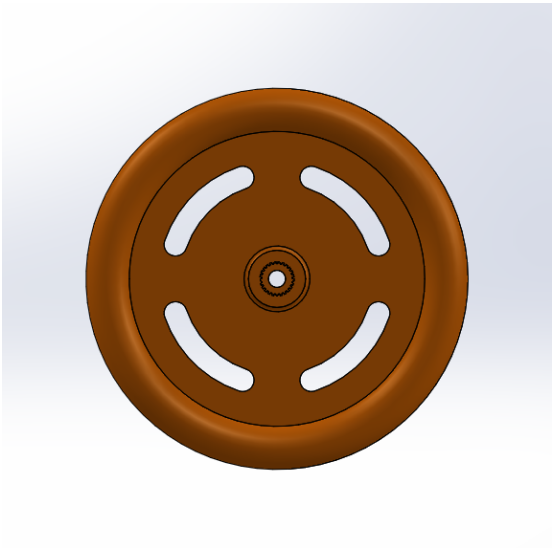

Figure 9 - Servo Wheel

The caster was designed to have a ping pong ball act as the wheel. The caster hole was sized as to have the ping pong ball fit snuggly but not too snuggly as to have it not move at all. 4 holes were added to a rim around the main cylinder to secure the caster to the underside of the frame using nuts and bolts.

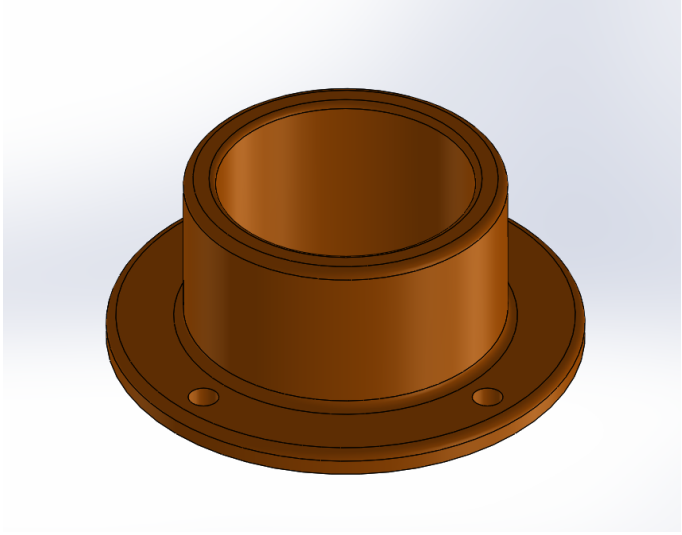

Figure 10 - Ping Pong Ball Caster

WIRELESS COMMUNICATION

Robot Transmitter Program

The transmitter code was inspired by the wireless communication program from lecture 12. The computer program sends to the Arduino the integer values (between 1000 and 2000) of the pulse widths of the servos controlling the wheels (pw_l and pw_r) and the servo directing the laser (pw_laser) as well as a integer value representing the status of the laser (1 for on and 0 for off) through serial communication. These variables are parameters from the set_inputs function, therefore those parameters need to be call by reference. The values of the parameters are stored in the outgoing serial buffer array after the set_inputs function is called in the main “while” loop.

The Arduino then receives the serial buffer and stores the variables into a structure called Data_Package, which has a maximum size of 32 bytes. Since 4 variables are being wirelessly transmitted, they must be sent as bytes. Before the data package is sent to the receiver, the variables are mapped from the range set in the computer program (1000-2000) to the range of a byte (0-255).

A function called reset_robot was devised as a backup in case of loss of connection during either serial communication or wireless communication. The function resets the values of the variables sent to the receiver to their default values, effectively stopping the robot in its tracks. The mid range value is sent as the pulse width of the servos (127 for a byte), giving no movement to the wheel servos and pointing the laser servo straight (i.e. 90 degrees). Furthermore, the laser is set to off by default. Due to the wheel servos being modified for continuous rotation, there was a slight deviation in their middle position, with the actual value for no movement being 129 (i.e. 91 degrees) for both wheels.

Robot Transmitter Circuit

The transmitter circuit simply features a nRF24 wireless transceiver module connected to an Arduino Nano. The wireless transmitter module features 8 pins, 7 of which will be used for our application: the GND, VCC, CE, CSN, SCK, MOSI, and MISO pins. The GND and VCC pins are connected to the GND and 3.3V pins on Arduino respectively. The SCK, MOSI and MISO pins are used for Serial Peripheral Interface (SPI) communication and must be connected to the SPI

pins on the Arduino Nano (pins 13, 11 and 12 respectively). The CE and CSN pins can be connected to any remaining digital pin and determine whether the module is in standby or active mode as well as if it is acting as a transmitter or receiver. [1]

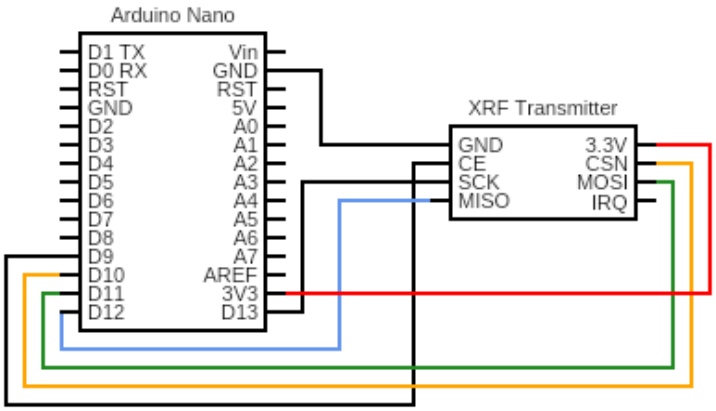

Figure 11 - Transmitter Circuit

Robot Receiver Program

On the receiver side, the Arduino receives the data package structure composed of the same variables as in the transmitter program. The received values are directly written to the appropriate outputs. The pulse widths sent to the servos are mapped from the range of a byte (0-255) to the range of the servo write function (0-180). Initially the writeMicroseconds was used to control the servos, since its range is the same as the one used in the computer program (1000-2000). However, it is common for servos to respond to values outside of this range due to some manufacturers not following the expected standard. [2] The value written to the led does not require mapping as it is either 0 or 1.

As stated in the transmitter program description, a failsafe was put in place in case of loss of communication. If more than 1 second has elapsed between receiving data packages, the values written to the robot are reset to their default values.

Robot Receiver Circuit

The receiver circuit features the same nRF24 module in the same configuration as in the transmitter circuit. Additionally, the 3 aforementioned servos are connected to PWM pins 3, 5 and 6 on the Arduino Nano. The led representing the laser is connected to digital pin 2. The servos are powered by an external 5V power supply while the arduino is powered by a 9V battery.

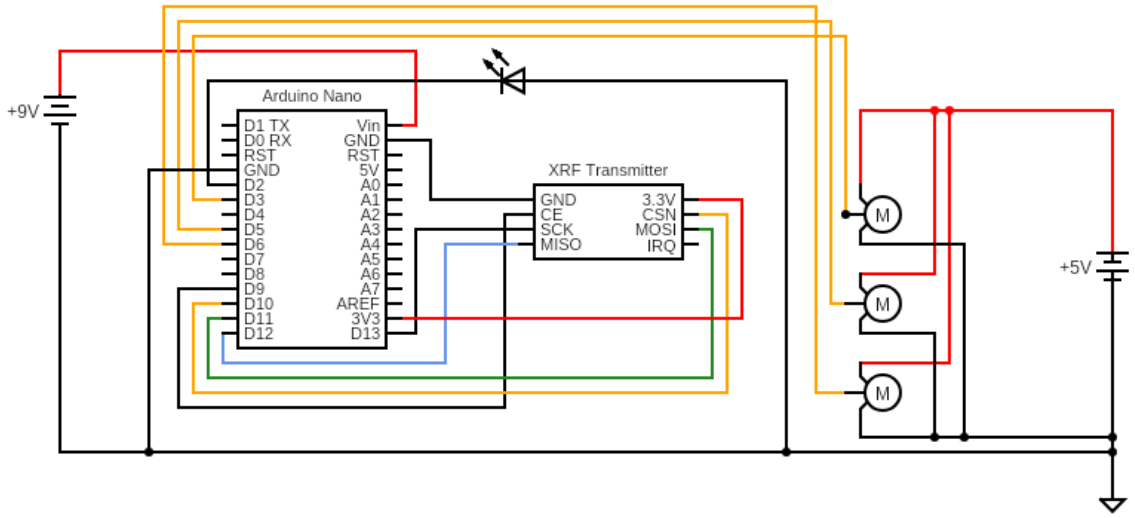

Figure 12 - Receiver Circuit

REFERENCES

[1] Dejan, "Arduino Wireless Communication – NRF24L01 Tutorial," How To Mechatronics, [Online]. Available: https://howtomechatronics.com/tutorials/arduino/arduino-wireless-communication-nrf24l01-tutorial/. [Accessed 7 April 2019].

[2] “Servo writeMicroseconds(),” *Arduino*. [Online]. Available: https://www.arduino.cc/en/Reference/ServoWriteMicroseconds. [Accessed: 03-May-2020].