01/13/20

Kelemen Mate
gr. 30432
mate.kelemen009@gmail.com

Obstacle Avoiding Robot Car

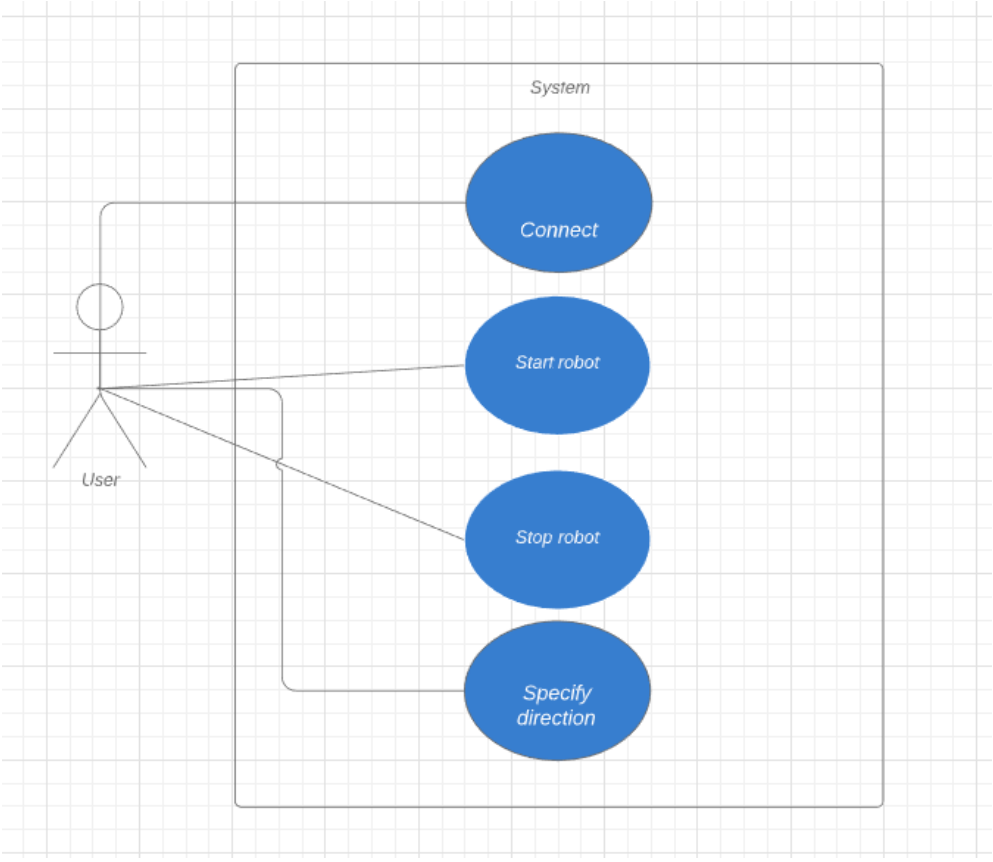
Table of Contents

1. Project purpose and objectives

The purpose of this project was to build a robot car and make it be able to be controlled by a user from a phone application. The other main function of the robot will be to avoid bumping into obstacles while being controlled by the user. The application that allows a user to control the robot is a custom android application that communicates the accelerometer data from the sensors of the phone via bluetooth to the robot. Therefore, by tilting the phone forwards and backwards, to the left and to the right, the user is able to tell the robot whether it is supposed to be moving forward or is it supposed to be steering in a direction. The obstacle avoidance mechanism that the robot will be implementing is a really basic one: whenever it sees an object close to it, it will temporarily disable the controls from the user, and turn in a direction where there are no objects close, so that when it gives back control to the user, the chances that it will run into an obstacle is minimal. The robot will steer smoothly, meaning that there will be no sudden changes in its direction. This will be done by slowly adjusting the rotation of each of its two motors to reach such values, that a forward vector specified by the inputs from the user can be approximated, and an error can be calculated. This way the motion of the robot is exprected to be relatively stable.

2. Analysis and theoretical foundation

Use cases



To reiterate the objectives, the user will be able to connect to the robot through bluetooth, to start and stop the robot, and to specify the direction that it ought to be moving towards. It is then the robot's responsibility to adjust its motors until the needed direction is reached, and to avoid obstacles.

Inputs of the robot and the phone application

The interface between the robot and the user will be an android phone. The data is needed to be sent as quick as possible, and as the quantity of one data packet is needed to be as short as possible, to ensure that all the data is processed quickly and instantaneously to the user. To provide a user-friendly simple interface, an application was developed for the sole purpose of sending accelerometer data to the robot to be processed further. The user is also able to connect to the robot through this application, by a single touch. As such, this application does nothing more than sending a continuous stream of two integers as inputs, after the user has successfully connected to the bluetooth sensor of the robot. The accelerometer data represents the pitch and the roll of the phone as two integers constrained within -100 and +100. The phone will send the robot one data packet every 10 milliseconds.

Algorithms

Distance calculation

It is trivial that distance calculation is needed in any form of obstacle avoidance. A cheap, easy to use solution is an ultrasonic sensor. An ultrasonic sensor can be used to measure speed through air, and not distance directly. The sensor works in the following way: it has a trigger transmitter and an echo receiver. We send out a pulse on the trigger, and after a while, if an object is close, it will reflect the emitted signal back to the sensor. The echo receiver senses this. Knowing the time it takes for the wave to come back since we emitted it, and the speed of sound through air, we can compute the distance in the following way:

*distance = (duration / 2) * speed of sound*

duration / 2 – we have to count the duration of the wave twice, because we have to wait until it reaches a surface to reflect off of, and once more for the reflection to come back to the sensor.

Motor adjustments

The key notion in the motor adjustments here, is the angular velocity around the OZ axis with which the robot potentially rotates. There are two states of the robot, in the case it did not sense any obstacle, and the user controls it: it either moves straight forward, or, if the user tilts the phone in either way, it moves to that direction. The robot will adjust the speed of its motors dynamically according to the *desired* angular velocity, calculated from the roll of the phone. This solves a few problems:

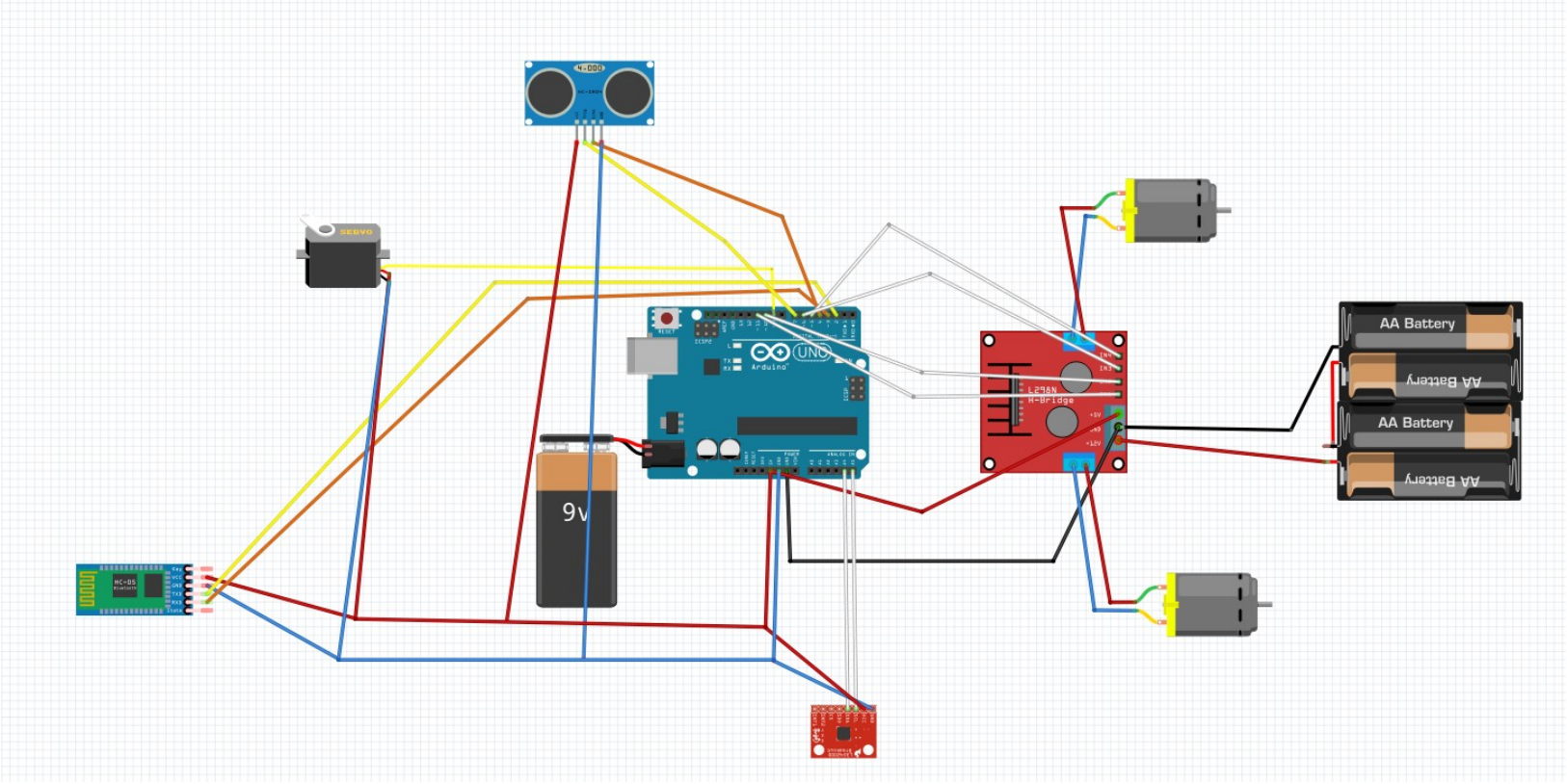
1. In case one motor is faster than the other one, even if we tell the robot to move straight, it will turn slightly to either left or right, depending on which one of the motors is stronger.
2. We can make the robot turn with a *specific* angular velocity, which makes controlling the robot less snappy and more fluid.

Using a gyroscope, the angular velocity around the OZ axis can be measured. Then we can calculate an error between the desired angular velocity and the current angular velocity, and we can adjust the motors based on this error. Knowing the maximum speed with which a motor can rotate, the algorithm is the following:

1. Calculate *error = target – current*
2. *If error < -threshold*
    1. *If left motor is not at maximum, increase its speed*
    2. *Else reduce the speed of the right motor*
3. *If error > threshold*
    - *If right motor is not at maximum, increase its speed*
    - *Else reduce the speed of the left motor*

3. Implementation

Hardware schematic



As the diagram shows, there are two different power sources to maximize the consumption of the motors and to power the Arduino Uno and the sensors connected to it. The two parts have a common ground.

Software design

Obstacle avoidance mechanism

The obstacle avoidance technique is the following:

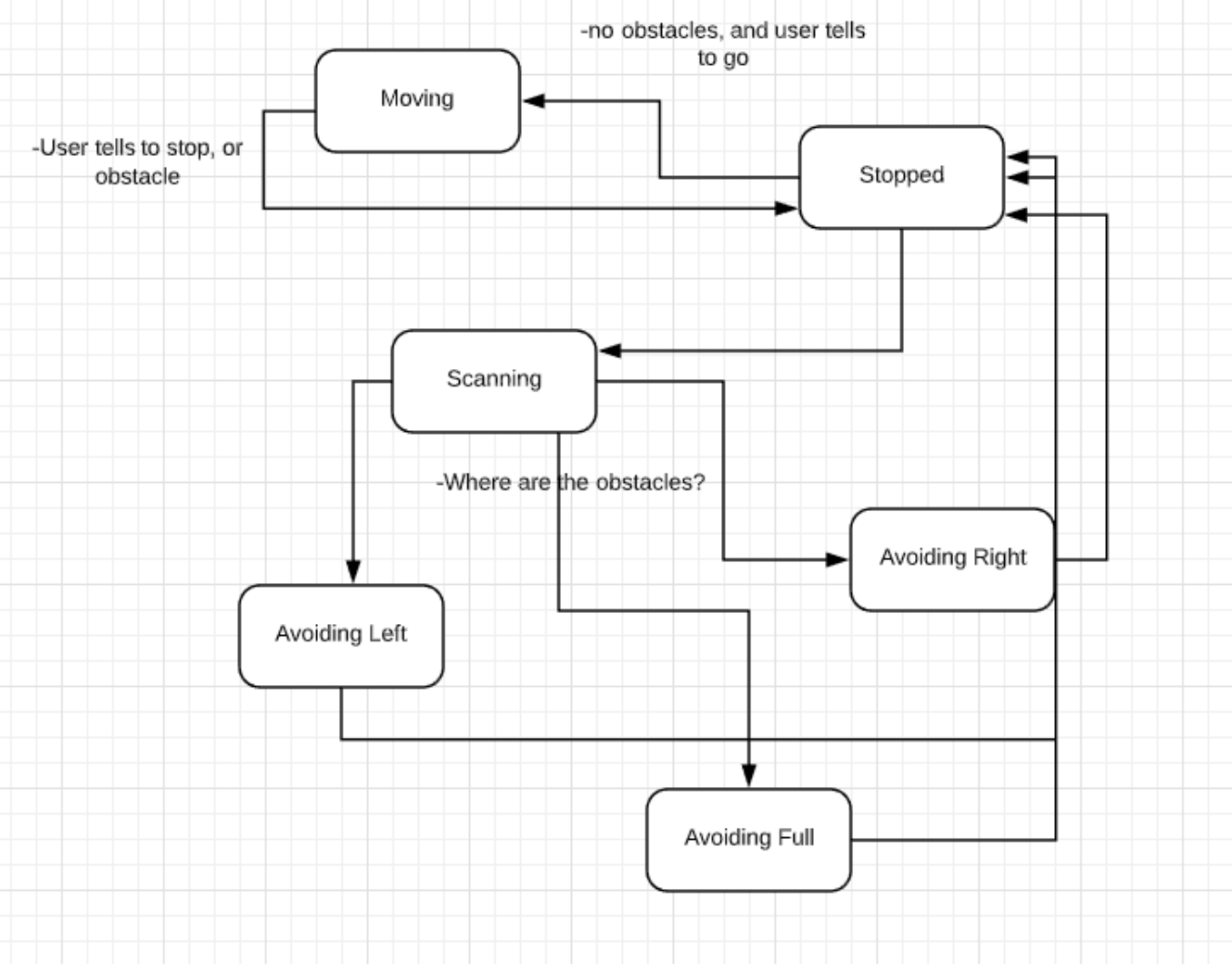
1. Whenever the robot senses an object close to it, it stops and gains control from the user
2. Scans for objects to the left and to the right
    - If there are objects on the left, turn approximately 90 degrees to right
    - If there are objects on the right, turn approximately 90 degrees to left

    - If there are objects both ways, turn around fully
- Give back control to the user

Approximation of the turn was done by calculating the yaw of the robot in every tick of the main loop, and saving it before turning.

State diagram

Knowing what the robot has to do, its behavior can be illustrated and implemented by the following state diagram:



4. Tests, results, conclusions

Tests were made in a home environment that is obstacle-rich, with an android phone. Connecting to the device was easy, and, as expected, the robot turns and adjusts itself smoothly. Tuning the parameters and debugging were also easy, with the help of printing to the serial and with a long usb cable. The robot also scans and turns away from obstacles as expected, although these parameters could be fine-tuned. A major problem that arose through the testing, is that when the robot finished scanning to left and right for obstacles, the bluetooth sensor turned off unexpectedly, and then turned on again. The robot also restarted in stopped state, and the deadly cycle continued: scan for obstacles and disconnect bluetooth. This suggests the likelihood of insufficiency of power, even with a 9V battery. Possible solution would be a rechargeable battery and redesigning the connections with the L293D board. On further inspection, sometimes the motors would move very sluggishly, and the bluetooth sensor would disconnect even in moderately short distances. With that being said, the robot still produced feasible results with a usb cable instead of a 9V battery.

5. Bibliography

https://www.arduino.cc/

https://www.youtube.com/watch?v=6F1B_N6LuKw&t=737s – ultrasonic sensor

https://www.youtube.com/watch?v=oQQpAACa3ac&t=1204s – robot car

6. Appendix – full source code

Robot

Sketch.ino

```
#include "Motor.h"

#include <Wire.h>

#include <MPU6050.h>

#include <Servo.h>

#include <TimerOne.h>


bool msg_in_end = false;
```

```
int accY, accX;
int zAxis = 0;

enum State {GOING, STOPPED, SCANNING, LEFT, RIGHT, FULL};

String inputString = "";
State currentState, prevState;

Motor m1, m2;
MPU6050 gyro;

int moveL, moveR;
float error;
unsigned long timer = 0;
float moveAmount = 0.0f;
float targetAngle = 0.0f;

float yaw = 0;
float tempYaw = 0;
float deltaYaw = 0;
const int adj_increment = 15;
float timeStep = 0.01f;

//ultrasonic
const int trig = A0;
```

```
const int echo = A1;

float targetZ = 0.0f;
int servoAngle = 85;

volatile bool is_avoiding = false;
int ledState = LOW;

Servo serv;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  m1.attach(6, 5);
  m2.attach(3, 11);

  currentState = STOPPED;

  while(!gyro.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_2G)){}

  while(!Serial){}

  gyro.calibrateGyro();
  gyro.setThreshold(3);

  inputString.reserve(200);
```

```
  moveL = 255;
  moveR = 255;
  error = 0;

  pinMode(trig, OUTPUT);
  pinMode(echo, INPUT);
  pinMode(LED_BUILTIN, OUTPUT);

  serv.attach(9);
  serv.write(85);

  delay(500);

  timer = millis();
}

void loop()
{
  zAxis = readGyro();

  if((millis() - timer) >= timeStep * 1000)
  {
    yaw += zAxis * timeStep;
    timer = millis();
  }
```

```
  processInputs();
  processState();
}

void serialEvent()
{
  while(Serial.available() > 0)
  {
    char inChar = (char)Serial.read();

    inputString += inChar;

    if(inChar == ';') msg_in_end = true;
  }
}

void processInputs()
{
  if(msg_in_end)
  {
    int ind = inputString.lastIndexOf(',');
    int semicolon = inputString.lastIndexOf(';');
    accX = inputString.substring(0, ind).toInt();
    accY = inputString.substring(ind + 1, semicolon).toInt();
    inputString = "";
    msg_in_end = false;
```

```
  }
}

void blinkLed()
{
  if(is_avoiding == true)
  {
    if(ledState == LOW)
    {
      ledState = HIGH;
    }

    else
    {
      ledState = LOW;
    }

    digitalWrite(LED_BUILTIN, ledState);
  }
}

void processState()
{
  if(currentState == STOPPED)
  {
      if(!is_avoiding) digitalWrite(LED_BUILTIN, HIGH);
```

```
    stopMotors();
    //start conditions trigger move forward
    if(metObstacle(15.0f)) currentState = SCANNING;

    else if(accY < -40)
    {
      currentState = GOING;
    }
  }

  else if(currentState == GOING)
  {
    if(!is_avoiding) digitalWrite(LED_BUILTIN, LOW);
    if(accX <= 40 && accX >= -40) accX = 0;
    if(accY >= -40) accY = 0;

    //targetAngle = map(accX, -100, 100, -70, 70);
    targetAngle = accX;

    adjust_motors(zAxis, targetAngle);

    moveMotors(moveL, moveR);

    if(accY == 0 || metObstacle(15.0f))
    {
      currentState = STOPPED;
```

```
    }
 }

 else if(currentState == SCANNING)
 {
    is_avoiding = true;
    //look both ways with the servo and check for distances
    serv.write(0);

    delay(1000);

    float distRight = calculateDistance();

    serv.write(180);
    delay(1000);

    float distLeft = calculateDistance();

    serv.write(85);
    delay(1000);

    if(distLeft <= 15.0f && distRight <= 15.0f)
    {
      currentState = FULL;
    }
```

```
    else if(distRight == -1)
    {
      currentState = RIGHT;
    }
    else if(distLeft == -1)
    {
      currentState = LEFT;
    }

    else
    {
      if(distLeft < distRight)
      {
        currentState = LEFT;
      }

      else
      {
        currentState = RIGHT;
      }
    }

    tempYaw = yaw;

    Serial.end();
  }
```

```
else if(currentState == LEFT)
{
   moveMotors(255, 0);

   if(abs(tempYaw - yaw) >= 40.0f)
   {
    tempYaw = 0;
    //resume
    currentState = GOING;
    is_avoiding = false;

    Serial.begin(9600);
   }
}

else if(currentState == RIGHT)
{
   moveMotors(0, 255);

   if(abs(tempYaw - yaw) >= 40.0f)
   {
    tempYaw = 0;
    currentState = GOING;

    is_avoiding = false;
```

```
      Serial.begin(9600);
    }
  }

  else if(currentState == FULL)
  {
    moveMotors(0, 255);

    if(abs(tempYaw - yaw) >= 80.0f)
    {
      tempYaw = 0;
      currentState = GOING;

      is_avoiding = false;
      Serial.begin(9600);
    }
  }
}

void adjust_motors(float gyro, float target)
{
  //right error = +, left = -
  error = gyro - target;

  if(error > 5)
  {
```

```
    if(moveR < 255 - adj_increment) moveR += adj_increment * abs(error);

    else if(moveL > adj_increment + 110) moveL -= adj_increment * abs(error);
   }

   else if(error < -5)
   {
    if(moveL < 255 - adj_increment) moveL += adj_increment * abs(error);

    else if(moveR > adj_increment + 110) moveR -= adj_increment * abs(error);
   }

   moveL = constrain(moveL, 110, 255);
   moveR = constrain(moveR, 110, 255);
}

int readGyro()
{
   return gyro.readNormalizeGyro().ZAxis;
}

int calculateDistance()
{
   float duration, distance;

   digitalWrite(trig, LOW);
```

```
  delayMicroseconds(2);
  digitalWrite(trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(trig, LOW);

  duration = pulseIn(echo, HIGH);
  distance = (duration / 2) * 0.0343f;

  if(distance >= 400 || distance <= 2)
  {
    distance = -1;
  }

  return distance;
}

bool metObstacle(float range)
{
  if (calculateDistance == -1) return false;
  return (calculateDistance() < range);
}

void stopMotors()
{
  m1.stop();
  m2.stop();
```

```
}

void moveMotors(int val1, int val2)
{
  m1.move(val1);
  m2.move(val2);
}
```

Motor.h

```
#ifndef MOTOR_H
#define MOTOR_H

#include <Arduino.h>

class Motor
{
  private:
    int pin1, pin2;

  public:
    void move(int);
    void attach(int, int);
    void detach();
    void stop();
};

#endif MOTOR_H
```

Motor.cpp

```
#include "Motor.h"

void Motor::attach(int p1, int p2)
{
  pin1 = p1;
  pin2 = p2;

  pinMode(pin1, OUTPUT);
  pinMode(pin2, OUTPUT);

  this->stop();
}

void Motor::move(int speed)
{
  if(speed != 0)
  {
    if(speed > 0)
    {
      analogWrite(pin1, speed);
      digitalWrite(pin2, LOW);
    }

    else
    {
```

```
    analogWrite(pin2, speed);
    digitalWrite(pin1, LOW);
   }
  }

  else stop();
}

void Motor::stop()
{
  digitalWrite(pin1, LOW);
  digitalWrite(pin2, LOW);
}

void Motor::detach()
{
  free(&pin1);
  free(&pin2);
}
```

Phone application

robot_tel:

```
import android.content.Intent;
import android.os.Bundle;
import ketai.net.bluetooth.*;
import ketai.sensors.*;
import java.nio.charset.Charset;
```

```
String msg;

KetaiSensor sensor;

float accX, accY, accZ;

long prevTime, currentTime;

long wait = 10l;
long time;

void setup()
{
  textAlign(CENTER, CENTER);
  textSize(18 * displayDensity);
  orientation(PORTRAIT);
  fullScreen();
  frameRate(60);
  ellipseMode(RADIUS);

  sensor = new KetaiSensor(this);
  sensor.start();
  bt.start();
  bt.getPairedDeviceNames();
```

```
  time = millis();
}

void draw()
{
  background(153, 224, 255);

  if(isConfiguring)
  {
    text("Touch to connect", width/2, height/2);
  }

  else
  {
    if(isConnecting)
    {
      if (bt.getConnectedDeviceNames().size() > 0)
      {
        isConnecting = false;
      }

      text("Connecting...", width/2, height/2);
    }

    else
    {
```

```
    if(millis() - time >= wait)
    {
      sendAcceleration();
      time = millis();
    }

    drawGUI();

    //sendAcceleration();
    text(accX, width/2, height/2 - 100);
    text(accY, width/2, height/2);

    if(bt.getConnectedDeviceNames().size() == 0)
    {
      isConfiguring = true;
    }
   }
  }
}

void mousePressed()
{
  if(isConfiguring && !isConnecting) connectToRobot();
}

void onAccelerometerEvent(float x, float y, float z)
```

```
{
  accX = x;
  accY = y;
  accZ = z;

  accX = constrain(accX, -4, 4);
  accY = constrain(accY, -4, 4);
  accZ = constrain(accZ, -4, 4);

  accX = map(accX, -4, 4, -1, 1);
  accY = map(accY, -4, 4, -1, 1);
  accZ = map(accZ, -4, 4, -1, 1);

  accX = floor(accX * 100.0f);
  accY = floor(accY * 100.0f);
  accZ = floor(accZ * 100.0f);
}

void sendAcceleration()
{
  String msg = (int)accX + "," + (int) accY + ";";

  sendToRobot(msg.getBytes());
}
```

bt_comm:

```
KetaiBluetooth bt;
```

```
boolean isConfiguring = true;
String lastMsg = "";
boolean isConnecting = false;
float timeScale = 0.001;

void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  bt = new KetaiBluetooth(this);
}

void onActivityResult(int requestCode, int resultCode, Intent data)
{
  bt.onActivityResult(requestCode, resultCode, data);
}

void onBluetoothDataEvent(String who, byte[] data)
{
  lastMsg = new String(data);
}

void sendToRobot(byte[] msg)
{
  bt.writeToDeviceName("HC-06", msg);
}
```

```
void connectToRobot()
{
  isConnecting = true;
  bt.connectToDeviceByName("HC-06");

  isConfiguring = false;
}
```