Dokumentacja projektu z przedmiotu Design Laboratory

Autorzy: Wojciech Nodzyński, Witold Klepek

1. **Temat:**

   „System reagowania na kolizję z wykorzystaniem Zumo Robota z myślą o walce z innym robotem”

2. **Założenia:**

   Projekt miał na celu opracowanie systemu odpowiedniego reagowania na kolizję robota w zależności od wykrytego miejsca zderzenia, który umożliwia zastosowanie bardziej optymalnej taktyki walki. Robot ucieka od przeciwnika lub podejmuje z nim walkę.

3. **Opis poszczególnych części kodu:**

3.1 Załączenie bibliotek oraz inicjalizacja zmiennych

Ta część zawiera załączenie bibliotek koniecznych m.in. do obsługi Zumo Robota, inicjalizacje obiektów odpowiadających za obsługę napędu, brzęczka czy guzika, a także zmienne przechowujące wartość Tresholdu do rozpoznania zderzenia, dane wejściowe odpowiadające za prędkość robota czy też zmienne symbolizujące parametry czasowe.

```
#include<ZumoShield.h>
#include<Wire.h>
ZumoBuzzer buzzer;
ZumoMotors motors;
Pushbutton button(ZUMO_BUTTON);

int XY_ACCELERATION_THRESHOLD = 16000;
const int motorSpeed = 120;
int RA_SIZE = 2;
unsigned long loop_start_time;
unsigned long contact_made_time;
int MIN_DELAY_BETWEEN_CONTACTS = 3000;
```

3.2 Szablon klasy RunningAverage

Załączony, aby obsługiwać w czasie średnią ostatnich pomiarów w akcelerometru. Ma to na celu zminimalizowanie wpływu „niepożądanych” odczytów, które mogłyby wpłynąć na detekcję kolizji w niewłaściwym momencie.

```
template <typename T>
class RunningAverage
{
  public:
    RunningAverage(void);
    RunningAverage(int);
    ~RunningAverage();
    void clear();
    void addValue(T);
    T getAverage() const;
    void fillValue(T, int);
  protected:
    int _size;
    int _cnt;
    int _idx;
    T _sum;
    T * _ar;
    static T zero;
};
```

```
T RunningAverage<T>::getAverage() const
{
  if (_cnt == 0) return zero; // NaN ?  math.h
  return _sum / _cnt;
}
```

3.3 Klasa akcelerometru oraz funkcje z nim związane

Zadaniem tej części kodu jest stworzenie wygodnego sposobu do odczytu wartości z osi X i Y akcelerometru oraz ich innych parametrów jak chociażby kąt wypadkowego wektora.

```
class Accelerometer : public ZumoIMU
{
  typedef struct acc_data_xy
  {
    unsigned long timestamp;
    int x;
    int y;
    float dir;
  } acc_data_xy;

  public:
    Accelerometer() : ra_x(RA_SIZE), ra_y(RA_SIZE) {};
    ~Accelerometer() {};
    void getLogHeader(void);
    void readAcceleration(unsigned long timestamp);
    float len_xy() const;
    float dir_xy() const;
    int x_avg(void) const;
    int y_avg(void) const;
    long ss_xy_avg(void) const;
    float dir_xy_avg(void) const;
  private:
    acc_data_xy last;
    RunningAverage<int> ra_x;
    RunningAverage<int> ra_y;
};
Accelerometer acc;
```

```
float Accelerometer::dir_xy_avg(void) const
{
  return atan2(static_cast<float>(x_avg()), static_cast<float>(y_avg())) * 180.0 / M_PI;
}
```

3.4 Funkcje obsługujące start robota

Funkcja CountDown() odpowiada za rozpoczęcie działania właściwego programu po wciśnięciu przycisku, zasygnalizowanie startu sygnałem z brzęczka oraz wyzerowaniem zmiennej ostatniego wykrytego kontaktu.

Funkcja setup() inicjalizuje magistrale I2C, konfiguruje komunikację z akcelerometrem oraz wywołuje wyżej wymienioną funkcję CountDown().

```
void CountDown()
{
  button.waitForButton();
  for(int i =0; i<3; i++)
  {
    delay(500);
    buzzer.playNote(NOTE_G(3), 250, 10);
    delay(500);
    buzzer.playNote(NOTE_G(4), 250, 10);
  }
  delay(1000);
  buzzer.playNote(NOTE_G(5), 600, 15);
  delay(500);

  contact_made_time = 0;

}
```

```
void setup()
{
   // Initialize the Wire library and join the I2C bus as a master
  Wire.begin();

  // Initialize accelerometer
  acc.init();
  acc.enableDefault();

#ifdef LOG_SERIAL
  Serial.begin(9600);
  acc.getLogHeader();
#endif
  randomSeed((unsigned int) millis());
  CountDown();

}
```

3.5 Funkcja loop() – ciągłość działania

Funkcja loop() zapewnia, że program po wciśnięciu przycisku będzie działał ciągle – jeśli wykryje kolizje wykona dalsze instrukcje obsługujące ją, natomiast domyślnie będzie poruszał się z zadaną prędkością. Ponadto funkcja zmienia Treshold na mniejszy po czasie 3 sekund od ostatniego kontaktu – ten zabieg eliminuje wywoływanie obsługi kolizji, gdy robot startuje.

```
void loop()
{
  if(button.isPressed())
  {
    motors.setSpeeds(0, 0);
    button.waitForRelease();
    CountDown();
  }


  loop_start_time = millis();
  if(loop_start_time>contact_made_time+3000)
    XY_ACCELERATION_THRESHOLD = 2400;
  acc.readAcceleration(loop_start_time);

  if(check_collision())
    {
      contact_detected();
    }
  else
  {
  motors.setSpeeds(motorSpeed, motorSpeed);
  }
}
```

3.6 Detekcja kolizji

Do obsługi tej części użyto dwóch funkcji. Check_collison() zwraca zmienną bool informującą o tym czy zdarzenie wystąpiło w zależności od zmierzonej wartości sumy kwadratów przyspieszenia w osiach X i Y względem wartości tresholdu podniesionej do kwadratu, a także w w zależności czy wystąpiło w odpowiednim odstępie czasowym od poprzedniego zdarzenia.

Procedura void contact_detected() aktualizue zmienną wystąpienia ostatniego zdarzenia oraz zmienną tresholdu (w celu wyeliminowania detekcji kolizji przy ponownym starcie robota). Ponadto wywołuje contact_response(), która zawiera właściwy opis zachowania się wobec kolizji.

```
bool check_collision()
{
  static long threshold_squared = (long) XY_ACCELERATION_THRESHOLD * (long) XY_ACCELERATION_THRESHOLD;
   return(acc.ss_xy_avg() >  threshold_squared)&& \
    (loop_start_time - contact_made_time > MIN_DELAY_BETWEEN_CONTACTS);

}
void contact_detected()
{

  contact_made_time = loop_start_time;
  contact_response();
  XY_ACCELERATION_THRESHOLD = 16000;

}
```

3.7 Obsługa kolizji

Na tą część kodu składają się contact_detected_where(), która zwraca wartość liczbową od 1 do 8 w zależności od tego, w którym z ośmiu zdefiniowanych sektorów zarejestrowano zderzenie, oraz contact_response(), która na podstawie wyboru dokonanego wcześniej wspomnianą funkcją, wykonuje instrukcje zachowania robota dopasowane do miejsca kolizji. Poniżej w drugim obrazku załączono instrukcje w jednym w wypadków, gdzie robot najpierw stara się obrócić, a po 200 ms zaczyna jechać w nowo wybraną stronę, aby uciec.

```
int contact_detected_where()
{
  if(acc.dir_xy()>=-110.0 && acc.dir_xy()<=-70.0)
    return 1;   //front
  else if(acc.dir_xy() > -70.0 && acc.dir_xy()< -20.0)
    return 2;    //front, right
  else if(acc.dir_xy() >= -20.0 && acc.dir_xy()<= 20.0)
    return 3;    //right
  else if(acc.dir_xy() > 20.0 && acc.dir_xy()< 70.0)
    return 4;      //back, right
  else if(acc.dir_xy() >= 70.0 && acc.dir_xy()<= 110.0)
    return 5;      //back
  else if(acc.dir_xy() > 110.0 && acc.dir_xy()< 160.0)
    return 6;      //back, left
  else if(acc.dir_xy() > 160.0 || acc.dir_xy()< -160.0)
    return 7;      //left
  else
    return 8;    //front, left
}
```

```
case 4:  //back, right
  motors.setSpeeds(motorSpeed*1.5 ,0);
  buzzer.playNote(NOTE_C(4), 400, 20);
  delay(200);
  motors.setSpeeds(motorSpeed, motorSpeed);
  buzzer.playNote(NOTE_G(4), 400, 20);
  delay(800);
  break;
```

4. **Źródła:**

https://playground.arduino.cc/Main/RunningAverage

https://github.com/pololu/zumo-shield-arduino-library

https://www.pololu.com/docs/0j57/all