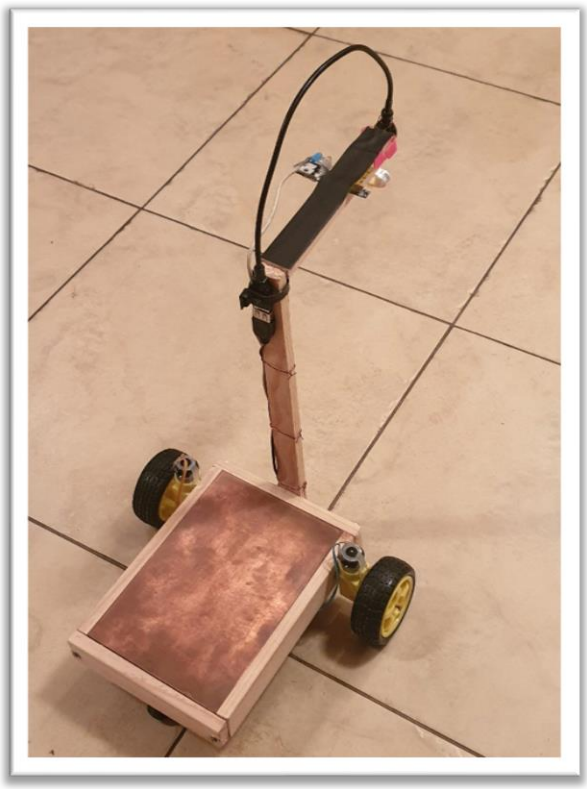
Line Follower ”Angler Fish”

Politechnika Poznańska WIiT

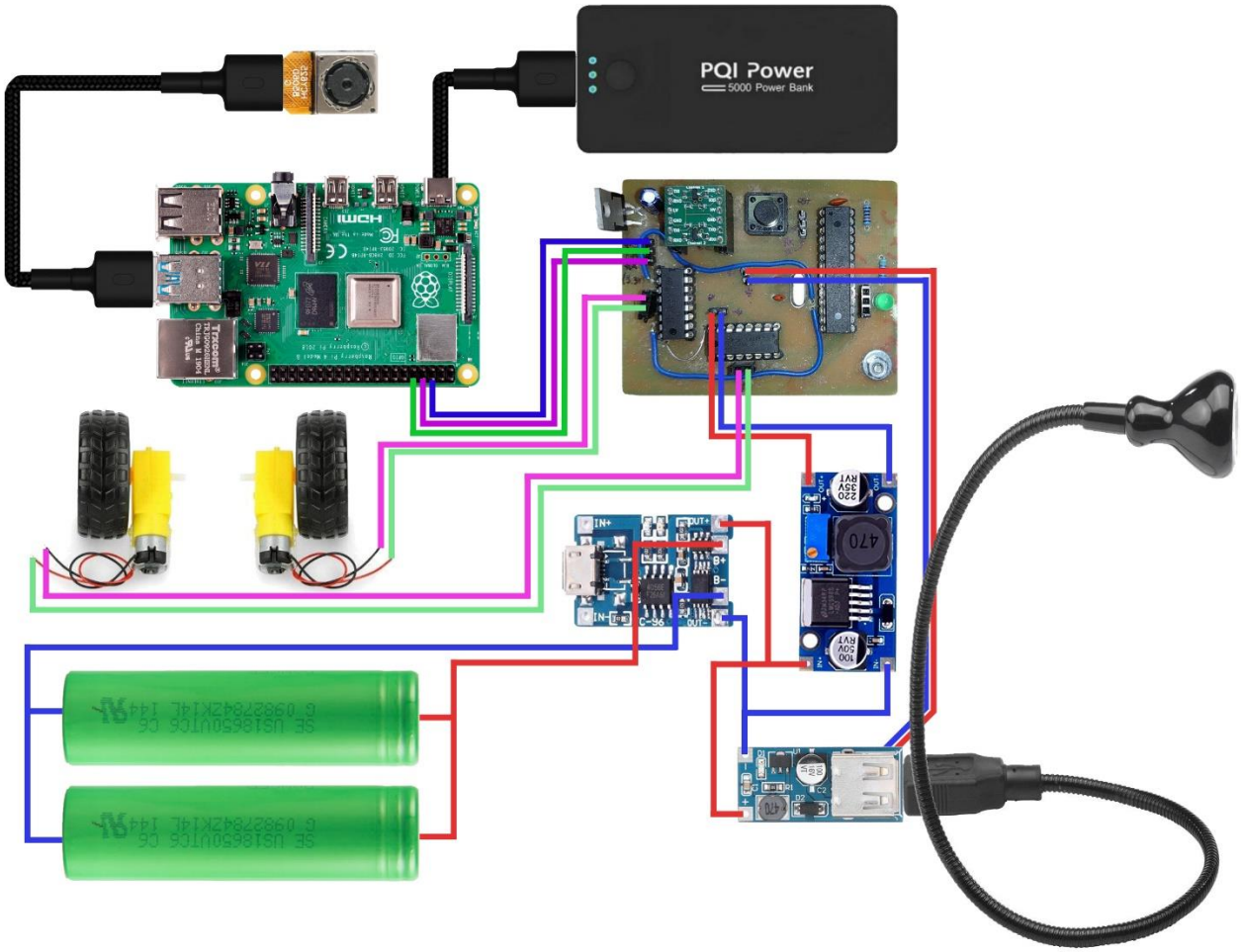
Architektura Systemów Komputerowych 2020



Zespół numer 30	
Osoba **[lider]**	Wykonane zadania
Roman Oberenkowski	• wykonanie ostatecznej obudowy • projekt i wytrawienie płytki • lutowanie • dopracowanie oprogramowania • DTR
Denys Hromniuk	• budowa prototypu • baza robota • ulepszanie oprogramowania • DTR
Daria Poda	• Organizacja pracy grupy • nagrywanie, montowanie wideoklipu (pomoc) • wykonanie ostatecznej obudowy • wytrawienie płytki
Wiktor Górczak	• główny program dla Raspberry • testowanie kamerki • program dla sterownika silników • DTR – opis programu
Pavlo Ravliv	• wideoklip • testowanie • dobieranie nastaw (współczynników – AP i KP)

1. Spis treści

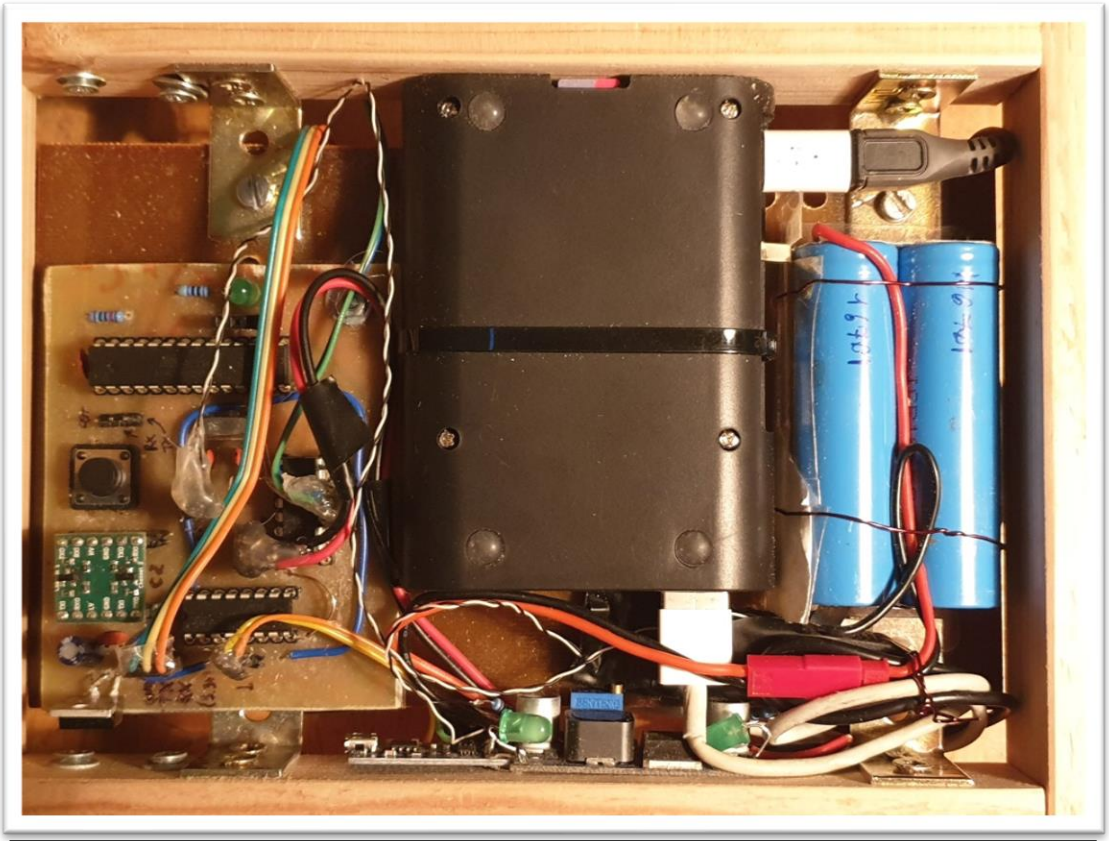
2. Ogólny schemat



3. Ciało robota

Konstrukcja ciała robota - ramka z drewnianych listewek. Kamera zainstalowana jest na stałe na drewnianym wysięgniku. Silniki po obu stronach przedniej części obudowy. Z tyłu - kółko obrotowe (360st.).

Pokrywa z laminatu pozwala na bardzo szybki dostęp do komponentów w środku obudowy:



Wnętrze naszego Line Followera: PCB, Raspberry Pi, ogniwa 18650 i reszta elektroniki; powerbank zamontowany jest od spodu

4. Zasilanie

Na początku robot działał bardzo krótko na jednym ładowaniu, więc podzieliliśmy zasilanie na dwie niezależne grupy: Raspberry Pi – pierwsza i sterownik+silniki+lampka – druga.

Wykorzystaliśmy:

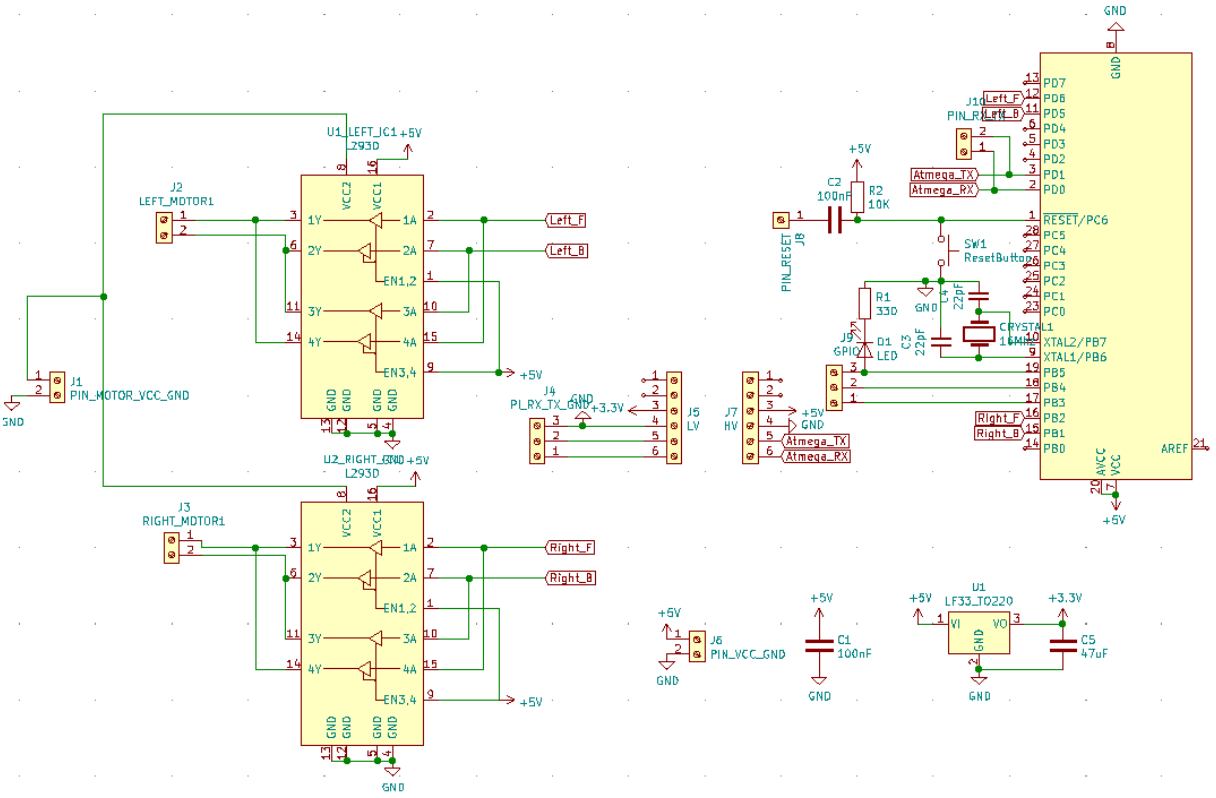
1. Dobry Powerbank (nie tani z marketu) dla zasilania Raspberry Pi (które wymaga 5V 2A)
2. Parę ogniw 18650 z
   a) modułem ładującym TP4056 (dodatkowo zabezpiecza ogniwa przed nadmiernym rozładowaniem
   b) przetwonicą step-up – zamienia napięcie z ~4V na większe (5V, dla silników; można je łatwo zwiększyć, lub zmniejszyć – potencjometrem na przetwornicy)
   c) modułem step-up booster DC-DC 5V – zasilanie logiki sterownika silników i lampki

5. Płytka drukowana - sterownik silników

Wykorzystaliśmy dwa układy L293D - jeden układ na jeden silnik. Można było jeden układ na dwa silniki, ale wtedy byliśmy na granicy specyfikacji (maksymalny prąd), więc wykorzystaliśmy dwa kanały jednego L293D na silnik.

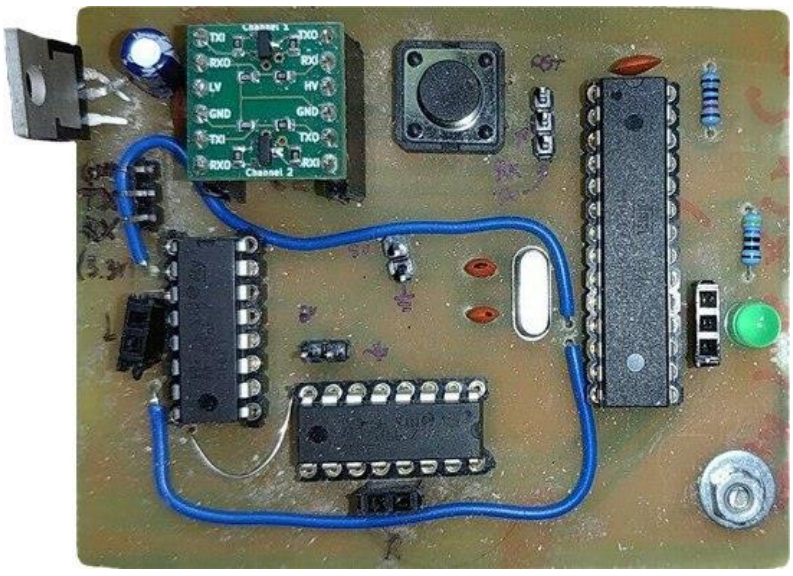
5.1. Projekt płytki drukowanej

Do zaprojektowania naszego układu wykorzystaliśmy darmowy program KiCad. Sercem naszego własnego systemu komputerowego jest Atmega328 z wgranym booloaderem Arduino, dzięki czemu można go programować przez Arduino IDE (potrzebna jest przejściówka USB-UART)



Projekt w programie KiCad

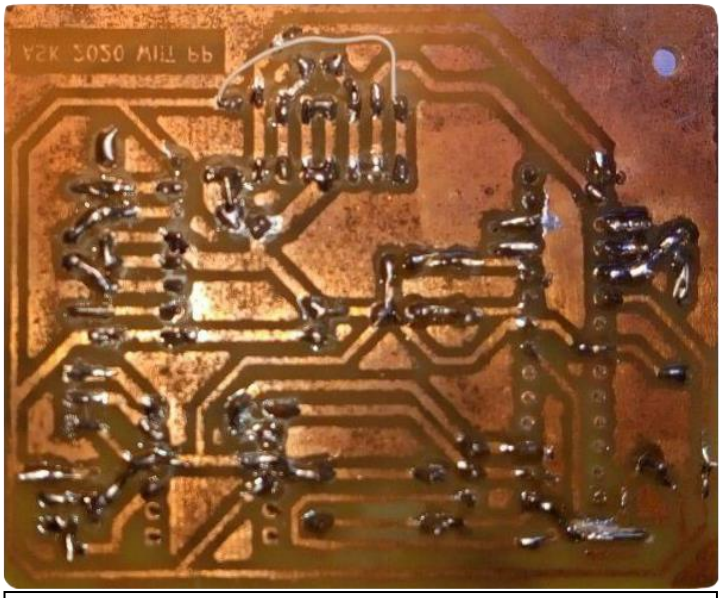
Komunikacja pomiędzy Raspberry Pi a Atmegą odbywa się przez UART, przez zintegrowany konwerter poziomów logicznych 3.3V – 5V na PCB. Raspberry pracuje na napięciu 3.3V, więc bezpośrednie połączenie z Atmegą, króra pracuje na 5V, mogłoby się skończyć spaleniem urządzenia.



Widok płytki z góry

Nie obyło się bez problemów - płytkę zaprojektowaliśmy dla regulatora LF33CV. Okazało się jednak, że został nam już tylko LM1117-3.3V. Regulatory te miały różne układy wyprowadzeń. Zamiast wytrawiać płytkę od nowa, odpowiednio wygięliśmy nóżki, żeby odpowiednio go podłączyć. Pilnowaliśmy, żeby nie doprowadzić do zwarcia. Stąd wygięte piny na zdjęciu płytki.

Wszystkie wyprowadzenia płytki są opisane na schemacie ogólnym (punkt 2). Wyjaśnienia wymaga jedynie złącze między przyciskiem i Atmegą – służy ono do programowania mikrokontrolera przez przejściówkę USB-UART. Z prawej strony znajduje się złącze 3-pinowe, które nie jest wykorzystywane, ale zostało dodane na wypadek przyszłych rozszerzeń.



Widok płytki z dołu

5.2. Kod sterownika

```
#define RIGHT_B 9
#define RIGHT_F 10
#define LEFT_B 5
#define LEFT_F 6
#define LEFT 0
#define RIGHT 1
#define FORWARD 1
#define BACKWARD 0
int tab[2][2];
void set_motor_speed(int lr,int speed);

void setup() {
  pinMode(RIGHT_B,OUTPUT);
  pinMode(RIGHT_F,OUTPUT);
  pinMode(LEFT_B,OUTPUT);
  pinMode(LEFT_F,OUTPUT);
  pinMode(13,OUTPUT);
  tab[LEFT][FORWARD]=LEFT_F;
  tab[LEFT][BACKWARD]=LEFT_B;
  tab[RIGHT][BACKWARD]=RIGHT_B;
  tab[RIGHT][FORWARD]=RIGHT_F;
  Serial.begin(9600);
}

void loop() {
   while (Serial.available() > 0) {
    int left = Serial.parseInt();
    int right = Serial.parseInt();

    if (Serial.read() == '\n') {
      PORTB ^= (1<<PB5);
      int lspeed = scale(left);
      int rspeed = scale(right);
      set_motor_speed(LEFT,lspeed);
      set_motor_speed(RIGHT,rspeed);
      }
   }
   delay(100);
}


int scale(int value){
  if(value==0)return 0;
  value=constrain(value,-100,100);
  if(value<0)value=map(value,-100,0,-254,-100);
  else value=map(value,0,100,100,254);
  value=constrain(value,-255,255);
  Serial.println(value);
  return value;
}
```

```
void set_motor_speed(int lr, int speed){
  if(speed>0){
    digitalWrite(tab[lr][BACKWARD],LOW);
    analogWrite(tab[lr][FORWARD],speed);
  }
  else if(speed<0){
    speed=-speed;
    digitalWrite(tab[lr][FORWARD],LOW);
    analogWrite(tab[lr][BACKWARD],speed);
  }
  else{
    digitalWrite(tab[lr][FORWARD],LOW);
    digitalWrite(tab[lr][BACKWARD],LOW);
  }
}
```

Prosty program napisany w Arduino IDE, który realizuje nasz sterownik silników. Odbiera prędkości silników przez UART i odpowiednio wysterowuje wyjścia. Funkcja scale jest odpowiedzialna za przeliczenie szybkości silników z zakresu <-100,100> na zakres -<255,-100> suma <100,255>

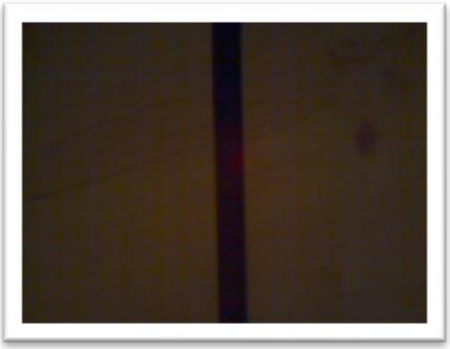
6. Kamerka

By ograniczyć koszty wykorzystaliśmy kamerkę ze starego laptopa, ale nie było to takie proste. Kamerka nie miała złącza USB, więc musieliśmy:

1. Dokonać inżynierii wstecznej – opracować pinout kamerki (pomocny film: https://www.youtube.com/watch?v=CouxmNqxO4A)
2. Zastosować regulator napięcia 3.3V, gdyż kamerka spaliłaby się na 5V.
3. Przylutować przewód ze złączem USB wg. zebranych wcześniej informacji
4. Przetestować i zabezpieczyć miejsce lutowania gorącym klejem

Podczas testów pojawiły się artefakty na obrazie z kamerki –zwiększenie parametru ilości klatek na sekundę z 15 do 20 rozwiązało problem. Prawdopodobnie ten model kamerki ma problemy z niższymi wartościami FPS.

Na początku robot był bardzo wrażliwy na warunki zewnętrzne – bez dodatkowego oświetlenia obraz był zbyt ciemny i pojawiało się dużo szumu na obrazie. Rozwiązaliśmy to przez dołożenie lampki doświetlającej od góry. Co spowodowało problem opisany poniżej.



Brak lampki, taśma izolacyjna

Lampka, taśma izolacyjna

Lampka, taśma malarska

Do tej pory wykorzystywaliśmy taśmę izolacyjną jako linię. Odbijała ona światło lampki jak na obrazku – kamera widziała odblask i nie rejestrowała linii w tym miejscu. Wymieniliśmy taśmę na granatową

matową taśmę malarską. Wymagało to dodania obsługi kolorów w programie, ale przyczyniło się to do dużo efektywniejszego śledzenia linii (mniej szumu).

Zbyt duża rozdzielczość powodowała duże opóźnienia w reakcji robota – obraz był zbyt obszerny, żeby można go było szybko przetworzyć. Zmniejszyliśmy rozdzielczość do zaledwie 160x120 pikseli, która jest jak najbardziej wystarczająca dla wykrywania konturów linii.

7. Raspberry

Przygotowaliśmy Raspberry do pracy zgodnie z listą kroków:

1. Nagranie systemu operacyjnego (Raspbian) na kartę MicroSD https://www.raspberrypi.org/downloads/raspberry-pi-os/
2. Konfiguracja i instalacja wymaganych pakietów i bibliotek (python i opencv2)
3. Testy kamerki + ustawienie parametrów (rodzielczość, fps)
4. Napisanie programu śledzącego linię

8. Właściwy program

Skrótowy (ideowy) opis działania naszego systemu komputerowego:

1. Automatyczne uruchamianie wirtualnego pulpitu i serwera vnc przy starcie Raspberry Pi (był problem że bez podłączonego monitora nie działał serwer graficzny i program nie mógł się uruchomić)
2. Automatyczne uruchamianie skryptu line_follower.py
3. Parametry kamerki – ustawienie rozdzielczości i ilości klatek na sekundę
4. Pobieranie ‘surowego obrazu’ z kamerki
5. Przepuszczanie przez filtry (dla niebieskiej taśmy – konwersja na HSV)
6. Erozja i potem „tycie” konturów obiektu celem wyeliminowania szumu
7. Wykrywanie konturów prostokąta na obrazie (najbliższego ostatnio wykrytemu w poprzedniej klatce)
8. Obliczenie odległości od linii i kąta do niej.
9. Obliczenie prędkości obu silników na podstawie danych z poprzedniego punktu
10. Wysłanie informacji do sterownika silników przez UART
11. Powrót do punktu czwartego.

9. Dokumentacja kodu źródłowego

W naszym programie korzystamy z biblioteki OpenCV do przetwarzania obrazu oraz WiringPI do połączenia ze sterownikiem silników. Wszystko zrealizowane jest w Pythonie. Tutaj znajduje się dokładny opis kodu programu.

9.1. Wstępna konfiguracja skryptu

Program rozpoczyna wykonywanie od ustawienia zmiennych konfiguracyjnych.

```
PROGRAM_NAME = "LF grupa 30"
resolution = (160,120)
CONST_FPS=20
cap = cv2.VideoCapture("/dev/video0")
cap.set(3, resolution[0])
cap.set(4, resolution[1])
cap.set(10,1.0)
set_fps(CONST_FPS)
clock=0
```

```
divider=CONST_FPS/10
#serial
wiringpi.wiringPiSetup()
serial = wiringpi.serialOpen('/dev/ttyS0',9600)
main_loop()
finish()
```

Najpierw ustawiamy nazwę programu i okna oraz parametry techniczne kamery, czyli jej rozdzielczość oraz oczekiwaną liczbę klatek na sekundę. Do zmiennej `cap` przekazujemy uchwyt do urządzenia biblioteki OpenCV. W zmiennej `divider` określamy z jaką częstotliwością mają być wysyłane dane na wyjście urządzenia. Korzystamy z biblioteki WiringPI celem połączenia wyjścia szeregowego, do którego podłączony jest nasz sterownik silników, z programem. Na końcu wchodzimy do pętli głównej programu, a gdy ta zostanie przerwana wykonywana jest funkcja sprzątająca.

9.2. Pętla główna programu

```
def main_loop():
   last_followed = int(resolution[0] / 2), int(resolution[1] / 2)
   while(True):
       ret, frame = cap.read()
       distance, angle, last_followed = process_frame(frame, resolution,
last_followed)
       temp_frame=cv2.resize(frame,(resolution[0]*2,2*resolution[1]))
       cv2.imshow(PROGRAM_NAME, temp_frame)
       key=cv2.waitKey(1) & 0xFF
       if key == ord('q'):
           break
       if key == ord(' '):
           manual_steering()
```

Inicjujemy zmienną last_followed środkiem naszego ekranu. Zmienna ta będzie nam potrzebna później, gdy będziemy musieli podjąć decyzję, którą linię wybrać, z wielu dostępnych do śledzenia tak, byśmy pozostali na naszej trasie. Jako, że jeszcze żadnej linii nie śledzimy - bierzemy po prostu punkt ze środka. Następnie wchodzimy do naszej pętli.

Z każdą iteracją pobieramy klatkę do zmiennej `frame`. Następnie wykonujemy metodę `process_frame`, która wyznacza dla nas:

- odległość od środka trasy (błąd),
- kąt pod jakim jedziemy w stosunku do trasy
- współrzędne prostokąta obejmującego obecnie śledzoną linię.

Wyświetlamy nasze wartości w oknie. Korzystamy ze zmiennej clock by ograniczyć częstotliwość wysyłania sygnałów do silników. Jeżeli nadszedł właściwy moment to przetwarzamy nasze zmienne w funkcji `process_angle_error`. Gdy nasz program wykona z ich pomocą właściwe operacje, wyświetlamy naszą klatkę na ekran i oczekujemy na wciśnięcie klawisza. Jeżeli jest to litera “q” to wychodzimy z pętli, jeżeli to spacja, to uruchamiamy ręczne sterowanie urządzeniem

9.3. Sterowanie ręczne

Zaimplementowaliśmy funkcję pozwalającą na sterowanie zdalne urządzeniem, ale jest ona bardzo prosta i absolutnie nie umożliwia płynnej jazdy. Była bardzo przydatna w diagnozowaniu problemów i testowaniu działania robota – żeby nie wstawać co chwila od komputera, gdy robot ma wjechać w ścianę, lub spaść ze schodów… Podczas sterowania ręcznego całkowicie wyłączone jest wykrywanie linii (nie pokazuje się nic poza surowym obrazem z kamerki). Ten tryb jest skrajnie prosty i pozwala tylko na:

- ‘cała naprzód!’
- obrót wokół własnej osi w lewo/prawo
- ‘cała wstecz!’

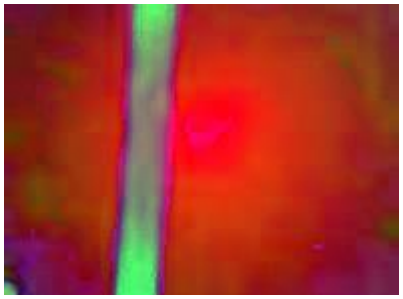
Sterowanie ręcznie nie było wykorzystywane podczas nagrywania.

```
def manual_steering():
    set_serial(0,0)
    manual_steering_speed=100
    while(True):
        ret, frame = cap.read()
        cv2.imshow(PROGRAM_NAME, frame)
        pre_process_frame(frame)
        key=cv2.waitKey(1) & 0xFF
        if key == ord(' '):
            set_serial(0,0)
        if key == ord('w'):
            set_serial(manual_steering_speed,manual_steering_speed)
        if key == ord('s'):
            set_serial(-manual_steering_speed,-manual_steering_speed)
        if key == ord('a'):
            set_serial(-manual_steering_speed*0.8,manual_steering_speed*0.8)
        if key == ord('d'):
            set_serial(manual_steering_speed*0.8,-manual_steering_speed*0.8)
        if key == ord('e'):
            break
        if key == ord('q'):
            finish()
```

9.4. Przetwarzanie pojedynczej klatki

Zanim przejdziemy do kodu zacznijmy od wizualizacji całego procesu

Etap 1 Surowy obraz z kamerki

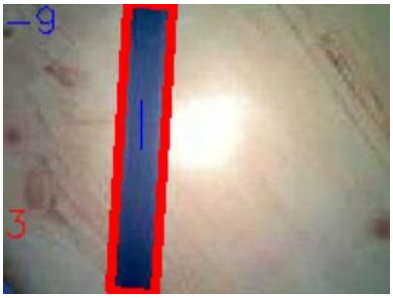


Etap 2 Konwersja na format HSV

Etap 3 Zaznaczenie pikseli mieszczących się w zadanym przedziale koloru

Etap 4 ‘Erozja’ kształtów

Etap 5 ‘Tycie’ kształtów



Etap 6 Wyświetlenie wyniku przetwarzania na tle surowego obrazu

Przetwarzanie pojedynczej klatki zaczyna się od wykonania funkcji pre_process_frame, która odpowiada za wykrywanie niebieskiej linii i usunięcie szumów.

```
def pre_process_frame(frame):
    #wykrywanie niebieskiej linii
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    lower_bound_blue = numpy.array([100,40,40])
    upper_bound_blue = numpy.array([140,255,255])
    followed_line = cv2.inRange(hsv, lower_bound_blue, upper_bound_blue)
    followed_line = remove_noise(followed_line,5,5)
    #odszymiony obraz
    return followed_line
```

Na początku konwertujemy kolory z formatu RGB do HSV, co umożliwi nam wygodniejsze posługiwanie się zakresem kolorów. Ustawiamy dolny i górny zakres koloru, które uznajemy za linię. Następnie odszumiamy obraz, by pozbyć się małych elementów, które mogą zostać omyłkowo wzięte za naszą linię.

```
def remove_noise(followed_line,errode_iterations,dilate_iterations):
    kernel = numpy.ones((3,3), numpy.uint8)
    followed_line = cv2.erode(followed_line, kernel, iterations=errode_iterations)
    followed_line = cv2.dilate(followed_line, kernel,
iterations=dilate_iterations)
    return followed_line
```

Usuwamy szumy przy pomocy opencv2. Funkcja errode ‘zjada’ część przefiltrowanego obrazu, więc małe kropeczki znikają. Niestety obraz teraz jest mniejszy, więc trzeba go ‘odbudować’ przez ‘tycie’ – funkcja dilate.

Bardzo dobra wizualizacja tego procesu znajduje się w dokumentacji biblioteki: Bezpośredni link

Właściwe przetwarzanie rozpoczyna się w funkcji process_frame.

```
def process_frame(frame, resolution, last_followed):
    x_res, y_res = resolution
    x_last, y_last = last_followed
    distance, angle = 0, 0
    followed_line=pre_process_frame(frame)
    image, contours, hierarchy = cv2.findContours(followed_line, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
```

Jako argumenty funkcja przyjmuje klatkę, rozdzielczość i współrzędne już śledzonej linii. Odpakowujemy je i inicjujemy zmienne z odległością. `Followed_line` to przetworzona przez OpenCV klatka, z której wyodrębnione są wszelkie elementy mieszczące się w naszej skali szarości. Przekazujemy tę zmienną do omówionej wyżej funkcji `pre_process_frame`. Następnie szukamy konturów ciemnych linii.

```
    if len(contours) > 0:
        if len(contours) == 1:
            selection = cv2.minAreaRect(contours[0])
        else:
            possible_selections = []
            off_bottom_selections = []

            for i in range(len(contours)):
                selection = cv2.minAreaRect(contours[i])
                (x, y), (width, height), angle = selection
                (x_bottom, y_bottom) = cv2.boxPoints(selection)[0] # pierwszy
```

```
wierzchołek jest zawsze najbliższy dolnej krawędzi
                if y_bottom >= y_res - 1:
                    last_followed_distance = ((x_last - x) ** 2 + (y_last - y) **
2) ** 0.5
                    off_bottom_selections.append((last_followed_distance, i))

                possible_selections.append((y_bottom, i))

            off_bottom_selections = sorted(off_bottom_selections)
            if len(off_bottom_selections) > 0:
                last_followed_distance, i = off_bottom_selections[0]
                selection = cv2.minAreaRect(contours[i])
            else:
                possible_selections = sorted(possible_selections)
                y_bottom, i = possible_selections[-1]

                selection = cv2.minAreaRect(contours[i])

        (x, y), (width, height), angle = selection

        if width*height>0.6*resolution[0]*resolution[1]:
            intersection=True
            angle=0
        else:
            intersection=False
            angle = int(compute_angle(width, height, angle))

        #wykrywanie skrętów pod kątem prostym i ewentualne korekty
        if width*height>0.2*resolution[0]*resolution[1] and not
width*height>0.6*resolution[0]*resolution[1]:
            if abs(angle)>70:
                if angle<0 and x>resolution[0]*0.5 and y>resolution[0]*0.5:
                    print("Right!")
                    angle=80
                elif angle>0 and x<resolution[0]*0.5 and y>resolution[0]*0.5:
                    print("Left!")
                    angle=-80
        middle = int(x_res / 2)
        distance = int(x - middle)
        box = cv2.boxPoints(selection)
        box = numpy.int0(box)

        box_color=(0, 0, 255) # ramka jest czerwona normalnie
        if intersection:
            box_color= (0, 255, 255) # a żółta na skrzyżowaniach
        cv2.drawContours(frame, [box], 0, box_color, 3)
        cv2.line(frame, (int(x), int(resolution[1]*0.6)), (int(x),
int(resolution[1]*0.4)), (255,0,0), 1)
        print_numbers(frame, angle, distance,intersection)
        process_angle_error(angle, 100*distance/(x_res/2.0))
return distance, angle, (x_last, y_last)
```

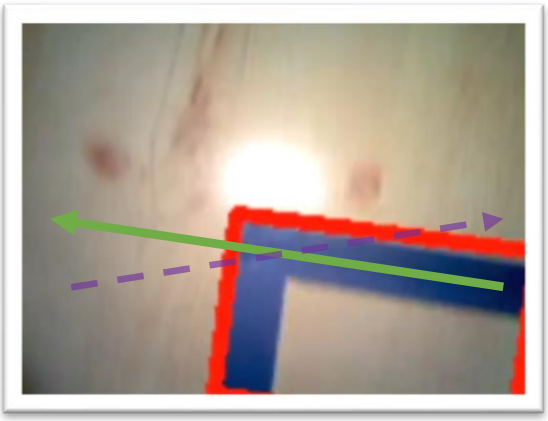
Jeżeli nie udało nam się znaleźć żadnego konturu, wychodzimy z funkcji bez żadnego przetwarzania. W przeciwnym wypadku sprawdzamy ile tych konturów zostało wykrytych. Jeżeli jeden, to sprawa jest prosta - jest tylko jedna linia, którą musimy śledzić, więc to jest nasz wybór. W przeciwnym wypadku musimy wybrać spośród kilku możliwości, które będziemy trzymać w tablicach `possible_selections` (na wszystkie możliwe linie) oraz `off_bottom_selections` (na te linie, które znajdują się bezpośrednio przy dolnej krawędzi klatki. Dokonujemy takiego podziału, ponieważ

wiemy na pewno, że w przypadku wielokrotnego wyboru, linia, na której już się znajdujemy, będzie sięgać dolnej krawędzi klatki, więc pole wyboru nam się zawęża. Jeżeli nie będzie takiej linii, bo na trasie znajduje się np. jakaś przerwa, wtedy musimy dokonać wyboru spośród wszelkich dostępnych opcji.

Iterujemy po wszystkich konturach. Z każdego konturu bierzemy pierwszy wierzchołek, który zawsze będzie tym najbliższym dolnej krawędzi. Jeżeli styka się on z dolną krawędzią, to obliczamy jego odległość od tego należącego do ostatnio śledzonej linii. Odległość ta jest potrzebna, bo właściwy wybór będzie tym najbliższym poprzedniemu.

Sortujemy nasze opcje tak, by wybrać najbliższą. Jeżeli jednak lista jest pusta, musimy wybrać ze wszystkich "ogólnych" opcji, które nie znajdują się przy dolnej krawędzi. Sortujemy i wybieramy tę opcję, która znajduje najbliżej dołu.

Odpakowujemy nasze wartości. Następnie sprawdzamy, czy nie dojechaliśmy do skrzyżowania, poprzez obliczenie pola naszego zaznaczenia. Jeżeli wynosi ono 60% całego obszaru roboczego, doszło do skrzyżowania, więc ustawiamy kąt skrętu na 0. W przeciwnym wypadku dedykowaną funkcją obliczamy właściwy kąt skrętu.



W następnym warunku sprawdzamy, czy wykryty został 'ostry' zakręt, czyli taki, który ma ponad 70 stopni i wykryty prostokąt linii znajduje się w lewym/prawym dolnym rogu. W takiej sytuacji, ustawiamy 'na sztywno' kąt 80 (lub odpowiednio -80) stopni, co pozwala skorygować okazjonalne niepoprawne zachowanie na skrętach pod kątem prostym. Przykład na rysunku obok – korygujemy sytuację, w której robot widząc taki obraz chciałby jechać w lewo (wzdłuż linii zaznaczonej zieloną strzałką). Programowo kąt jest zmieniany, a co za tym idzie zwrot także. Ostateczny efekt jest taki, że robot jedzie za 'sztuczną' linią zaznaczoną przez fioletową przerywaną strzałkę. Czyli gwałtownie skręca w prawo, o co nam chodziło. W kolejnej sekundzie mechanizm ten się wyłączą (nie są spełnione oba warunki) i wszystko odbywa się w normalnym trybie.

Wyświetlamy również stosowny komunikat na konsoli. Dalej obliczamy odległość od środka obszaru roboczego oraz wyświetlamy wszystkie zaznaczenia. Wyświetlamy wyniki na ekranie i obliczone współczynniki, przekazujemy do funkcji `process_angle_error`, która obliczy odpowiednie wartości dla silników.

9.5. Obliczanie kąta

```
def compute_angle(width, height, angle):
    if angle < -90 or (width > height and angle < 0):
        return 90 + angle
    if width < height and angle > 0:
        return (90 - angle) * -1
    return angle
```

Proste obliczenia trygonometryczne

9.6. Przetwarzanie odległości i kąta

```
def process_angle_error(angle,error):
    global clock
    clock+=1
    if(clock>divider):
        clock=0
        KP=1.25
        if abs(angle)<15:
            KP=1.75
        AP=1.25
        speed=100
        steering = error * KP + angle * AP
        if steering == 0:
            port1=speed
            port2=speed
        else:
            if steering > 0:
                steering = 100 - steering
                port2=speed
                port1=speed*steering/100
            else:
                steering = steering * -1
                steering = 100 - steering;
                port1=speed
                port2=speed*steering/100
        max_backward_speed=50
        if port1<-max_backward_speed:
            port1=-max_backward_speed #right
        if port2<-max_backward_speed:
            port2=-max_backward_speed #left
        set_serial(port2,port1)
```

Współczynniki KP i AP wyznaczają, jaki wpływ na skręt ma mieć kąt, a jaki odległość od środka. Jeżeli zmienna `steering` ma wartość 0, to jedziemy prosto - na porcie zarówno lewej, jak i prawej strony ustawiamy tę samą prędkość. W przeciwnym wypadku spowalniamy jedną ze stron w zależności od wartości współczynnika. Ograniczamy maksymalną prędkość jazdy do tyłu, żeby uniknąć zbyt szybkiego obrotu i stracenia linii z pola widzenia. Ostatnia linijka wysyła komendę do sterownika ustawiającą prędkość obu silników.

9.7. Informacje graficzne

```
def print_numbers(frame, angle, distance,intersection):
    angle_pos=(0,97*resolution[1]//100)
    error_pos=(0,12*resolution[1]//100)
    inters_pos=(93*resolution[0]//100,97*resolution[1]//100)
    cv2.putText(frame, str(angle), angle_pos, cv2.FONT_HERSHEY_SIMPLEX, 0.6,
(0,0,255), 1)
    cv2.putText(frame, str(distance), error_pos, cv2.FONT_HERSHEY_SIMPLEX, 0.6,
(255,0,0), 1)
    if(intersection):
       cv2.putText(frame, "S", inters_pos, cv2.FONT_HERSHEY_SIMPLEX, 0.6,
(0,255,255), 2)
```

Ta funkcja jest odpowiedzialna za wyświetlanie odpowiednich informacji podczas działania programu kąta na czerwono i odległości od środka na niebiesko. Dodatkowo, kiedy zostanie wykryte skrzyżowania w prawym dolnym rogu pokazuje się żółta litera ‘S’.

10. Kinematyka i silniki

W prototypie jeden silnik był szybszy od drugiego – wymagana była kalibracja w programie i ustawiania wartości prędkości silnika lewego zawsze na 15% więcej.

Silniki startowały dopiero od jakiegoś wypełnienia PWM[1] (około połowy) więc sterownik silników zmienia skalę. Tak, że 0% = zero wypełnienia, a 1% = 100, 100%=255 wypełnienia.

Przy przejściu z prototypu na finalną wersję zmieniliśmy ułożenie silników (o 90stopni), co ogólnie zmniejszyło prędkość silników na danym napięciu (prawdopodobny powód: większe tarcie przy takim ustawieniu silników). Co ciekawe taka zmiana ustawienia naprawiła problem, gdzie jeden silnik był szybszy od drugiego – w finalnym programie nie ma 15-procentowego przyśpieszenia lewego silnika.

Zastosowaliśmy mostki H, więc nasz robot może wykorzystywać koła do jazdy i do przodu i do tyłu.

11. Koszty/spis części

Prawie wszystkie części były pozyskane z ‘garażowych zasobów’.

Kamerka do robota została wymontowana ze starego zepsutego laptopa HP 2710p (~2008r)

Para ogniw 18650 z baterii od innego laptopa - zwykle w baterii od laptopa psuje się jedna para ogniw, dwie pozostałe są jak najbardziej sprawne.

Gdyby ktoś chciał wykonać podobny projekt we własnym zakresie tak przedstawiałby się lista części:

[1] PWM – pulse with modulation

Komponent	Uwagi
Raspberry pi 4 + karta MicroSD	Można wykorzystać starszy model (Pi2 lub Pi3). Pi1 będzie raczej już zbyt wolne.
Ogniwo 18650	Wymontowane z baterii od laptopa
Kamerka	Wymontowana z laptopa
Silniki	Najzwyklejsze i najtańsze wystarczą do naszych celów
Kółko tylne 360st.	Można nasmarować przed użyciem
Lampka LED na USB	Lub inne źródło światła np. latarka
Powerbank + przewód USB-A USB-C	5V co najmniej 2A (czyli tanie z marketu odpadają)
Przetwornica Step-Up	Regulowane wyjście pozwala łatwo przyśpieszać i zwalniać silniki w zależności od potrzeb
Step-up booster 5V USB	Generuje stabilne i dokładne 5V
Moduł ładujący i zapezpieczający 18650	Dzięki prostemu i łatwemu w obsłudzie ładowaniu nie musimy kupować wielu baterii jednorazowych
Atmega328	Alternatywnie arduino (wtedy płytka tylko z L293D)
L293D	Dwie sztuki
Zielona dioda	
Przycisk	
Kondensatory	Ceramiczne do rezonatora kwarcowego i stabilizacji napięcia + elektrolityczne do stabilizacji napięcia i podłączenia złącza resetu Atmegi
Rezonator kwarcowy 16MHz	By można było wykorzystać Arduino IDE Atmega musi być taktowana 16MHz
Laminat i wytrawiacz	Dodatkowy laminat jako część obudowy
Narzędzia	Lutownica, wiertarka, papier ścierny, rozpuszczalnik, gorący klej, marker, żelazko, komputer itp.
Przewody różnego rodzaju	Dostępne wszędzie
Wtyki i gniazda goldpin	Bardzo ułatwia to prototypowanie, gdy można wszystko łatwo rozłączyć na części składowe
Rezystory	Pull-up i do diod[2]
Drewniane listewki	
Śubki/Wkręty	Raczej mniejsze
Opaski zaciskowe	Obowiązkowe wyposażenie każdego warsztatu
Taśma izolacyjna i malarska	“Na taśmie cywilizację zbudowano” autor nieznany
Drut miedziany	Zawsze przydatny
Regulator 3.3V	Jeden dla kamerki, drugi na płytkę
Konwerter poziomów logicznych	Potrzebny do połączenia UART między urządzeniami działającymi na różnych napięciach (5V i 3.3V)
Listwy żeńskie pinowe 2mm	Jeśli układ się spali, to nie trzeba wylutowywać – można delikatnie wyciągnąć, działa jak gniazdo.

Prawie wszystkie części/narzędzia były dostępne w garażach/warsztatach członków zespołu. Z narzędzi musieliśmy zamówić tylko malutkie wiertła i wytrawiacz do płytki drukowanej. Był to marginalny koszt (25zł).

[2] Poprawna odmiana – diod (NIE diód) PWN link

12. Geneza nazwy Angler Fish

Lampka u góry robota jest bardzo podobna do źródła światła, którym ryby głębinowe wabią swoje ofiary. Z uwagi na to podobieństwo (wizualne oczywiście) – zdecydowaliśmy się tak nazwać naszego robota.