

PROJEKT
STEROWNIKI ROBOTÓW

Dokumentacja

Robot balansujący piłką
K.U.L.A.

Skład grupy:
Michał TRELA, 259312
Jan MASŁOWSKI, 258962

Termin: wtTN19

Prowadzący:
dr inż. Wojciech DOMSKI

Spis treści

1 Opis projektu	2
2 Założenia projektowe	2
2.1 Konfiguracja pinów	2
2.1.1 Raspberry Pi 4	2
2.1.2 Arduino Uno	2
2.2 CSI	2
2.3 Opis konstrukcji	3
2.4 Lokalizacja piłki	3
2.5 Balansowanie piłką	3
3 Konstrukcja mechaniczna	3
4 Opis działania programu	7
4.1 Kalibracja kamery	7
4.2 Wyznaczanie zakresu koloru piłeczki w modelu HSV	8
4.3 Detekcja markerów	9
4.4 Detekcja piłki	10
4.5 Regulator PID	11
4.6 Sterowanie serwomechanizmami	12
4.7 Główny program	13
5 Harmonogram pracy	16
5.1 Podział pracy	16
5.2 Kamienie milowe	16
6 Dokumentacja projektu	17
Bibliografia	18

1 Opis projektu

Projekt ma na celu stworzenie robota balansującego piłkę w dwóch osiach. Lokalizacja piłki na platformie w odniesieniu do centrum będzie odbywać się za pomocą systemu wizyjnego a sterowanie wychyleniem platformy będzie się odbywać za pomocą regulatora PID, który wyznaczać będzie pozycję serwomechanizmów w celu utrzymania piłeczki na środku platformy. Przetwarzanie obrazu oraz regulator PID zostaną zaimplementowane na minikomputerze Raspberry Pi 4 a sterowanie serwomechanizmami będzie wykonywane pośrednio poprzez komunikację UART z Arduino Uno.

2 Założenia projektowe

2.1 Konfiguracja pinów

2.1.1 Raspberry Pi 4

Tabela 1: Konfiguracja pinów mikrokontrolera

Numer pinu	PIN	Tryb pracy	Funkcja/etykieta
32	GPIO14	UART_TXD0	TRANSMIT
33	GPIO15	UART_RXD0	RECEIVE

2.1.2 Arduino Uno

Tabela 2: Konfiguracja pinów mikrokontrolera

Numer pinu	PIN	Tryb pracy	Funkcja/etykieta
0	PD0	RX	RECEIVE
1	PD1	TX	TRANSMIT
10	PB2	PWM	PWM_1
11	PD3	PWM	PWM_2

2.2 CSI

Do obsługi kamery zostanie wykorzystany zostanie wbudowany interfejs komunikacyjny CSI (Camera Serial Interface). Konfiguracja kamery będzie odbywać się za pomocą API - PiCamera.

Tabela 3: Konfiguracja pinów portu CSI

PIN	Nazwa	Opis
1	GND	Ground
2	CAM_D0_N	MIPI Data Lane 0 Negative
3	CAM_D0_P	MIPI Data Lane 0 Positive
4	GND	Ground
5	CAM_D1_N	MIPI Data Lane 1 Negative
6	CAM_D1_P	MIPI Data Lane 1 Positive
7	GND	Ground
8	CAM_CK_N	MIPI Clock Lane Negative
9	CAM_CK_P	MIPI Clock Lane Positive
10	GND	Ground
11	CAM_IO0	Power Enable
12	CAM_IO1	LED Indicator
13	CAM_SCL	I2C SCL
14	CAM_SDA	I2C SDA
15	CAM_3V3	3.3V Power Input

2.3 Opis konstrukcji

Komputerowy model został wykonany korzystając z technologii CAD. Uchwyty na platformę, kamerę, serwomechanizmy oraz elektronikę został wykonany za pomocą druku 3D. Wysięgnik na kamerę został wykonany z profili aluminiowych. Całość została przymocowana do drewnianej deski służącej jako podstawa. Płyta akrylowa została użyta jako platforma, na której piłka jest balansowana.

2.4 Lokalizacja piłki

Lokalizacja piłki odbywa się poprzez analizę obrazu otrzymanego z kamery korzystając z biblioteki OpenCV. Środek platformy jest wyznaczany przez markery ArUco umieszczone w rogach platformy. Miejsce przecięcia przekątnych poprowadzonych z skrajnych punktów wyznaczają środek. Następnie wyznaczana jest pozycja piłki na obrazie oraz wyznaczany jest wektor przemieszczenia piłki względem środka platformy. W celu lepszej stabilizacji platformy balansującej użyto gumowych odciągów.

2.5 Balansowanie piłką

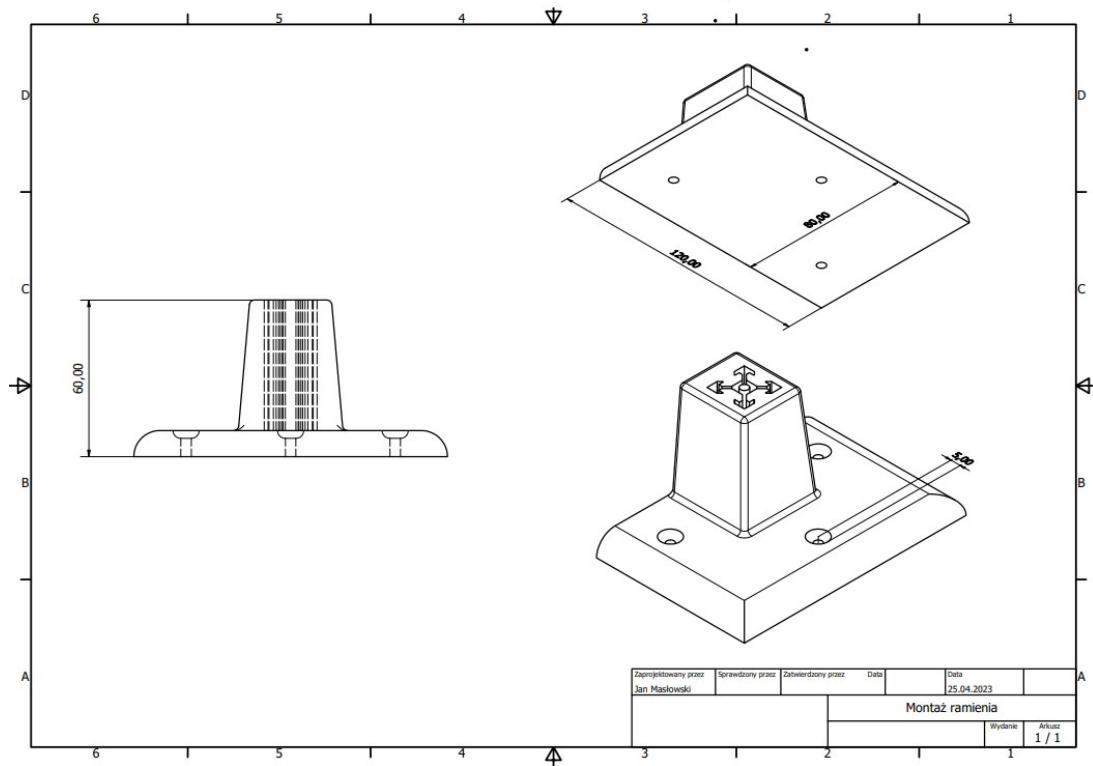
Po wyznaczeniu wektora przemieszczenia, błąd jako odległość w osi od środka jest przekazywany na wejście regulatora PID. Wychylenie w obu osiach jest regulowane za pomocą dwóch regulatorów, po jednym na oś. Działają one niezależnie od siebie.

3 Konstrukcja mechaniczna

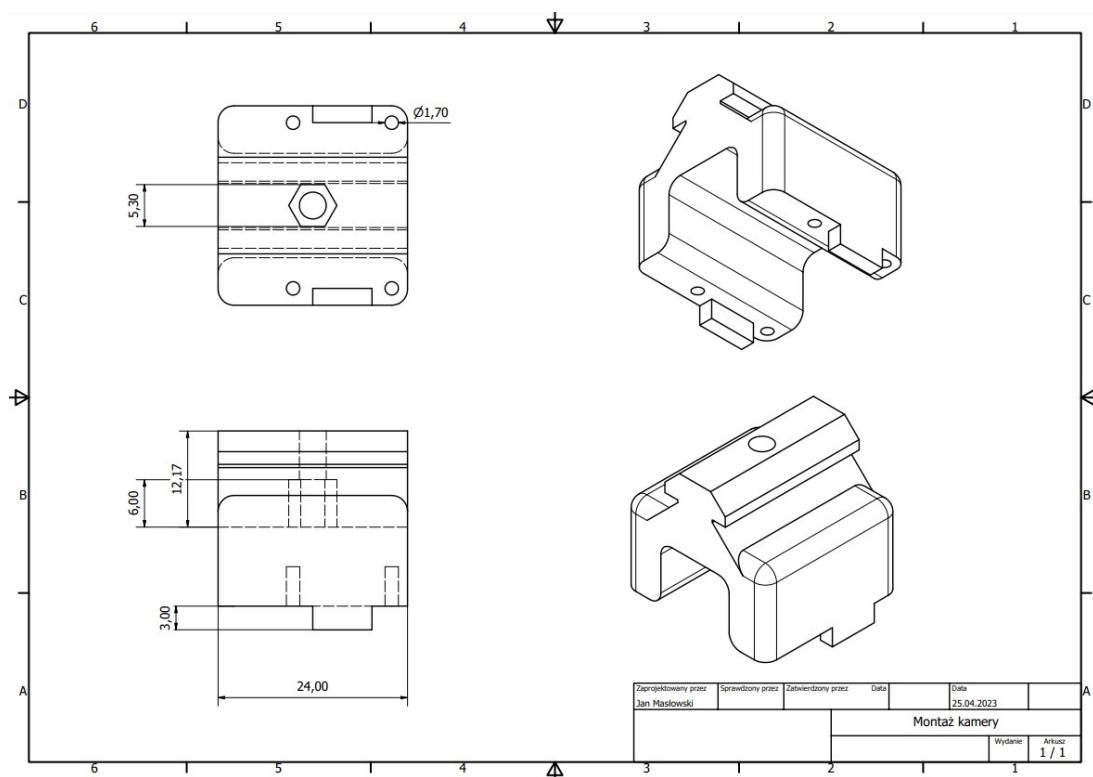
Platforma balansująca składa się z następujących elementów:

- Podstawa ze sklejki o wymiarach 29 x 36.5 x 1.8 cm
- Dwóch profili aluminiowych V2020 o wymiarach 50 i 25 cm
- Łącznika profili aluminiowych
- 11 elementów stworzonych w technologii druku 3D
- Płyta akrylowa o wymiarach 30 x 30 cm
- Śrub, wkrętów i nakrętek

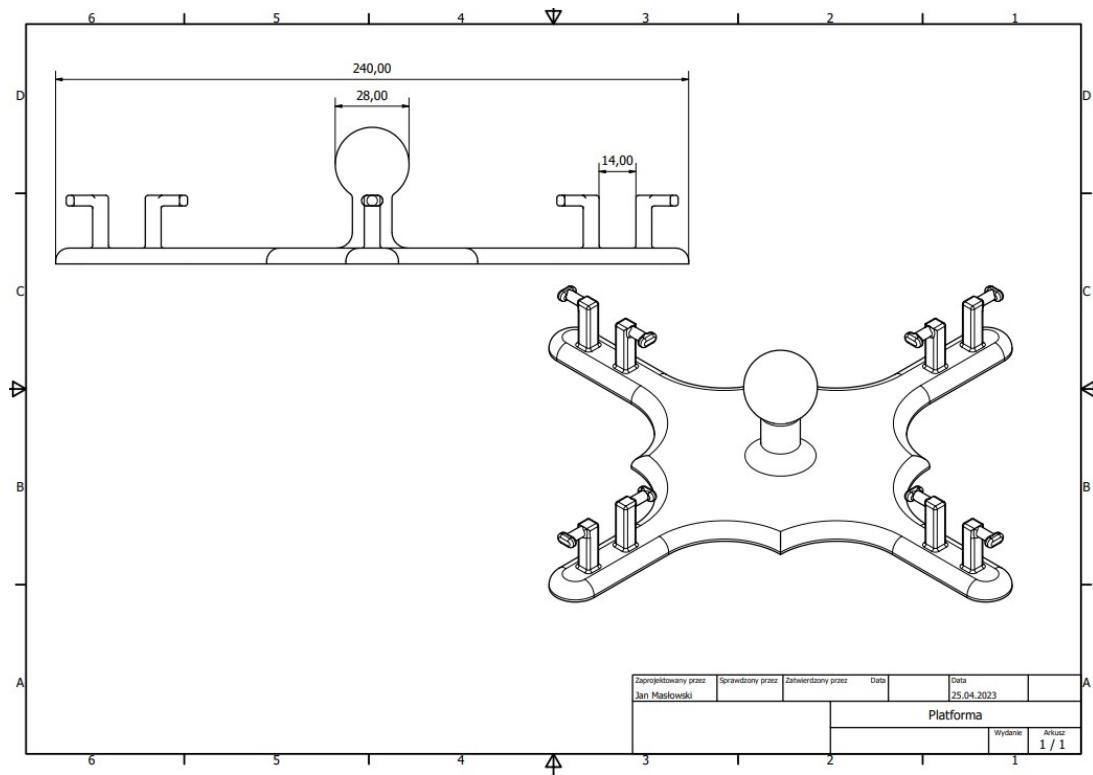
Wybrane rysunki techniczne elementów drukowanych:



Rysunek 1: Podstawa ramienia kamery.

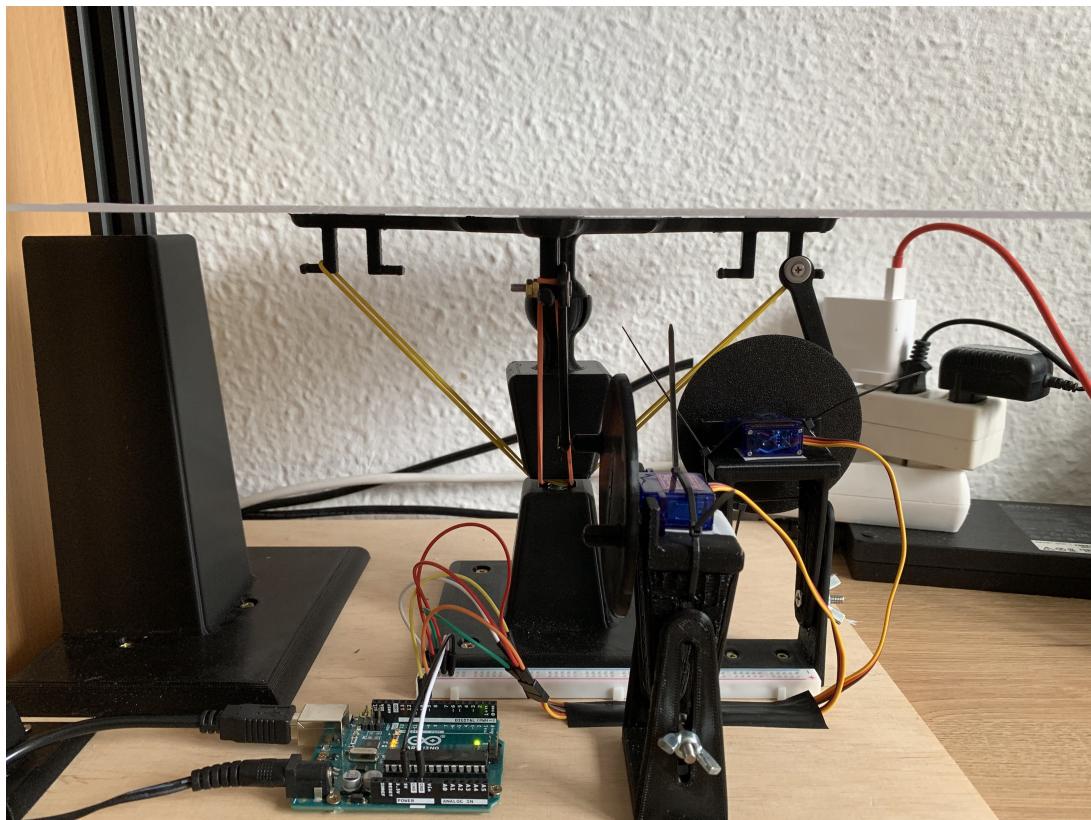


Rysunek 2: Uchwyt kamery.

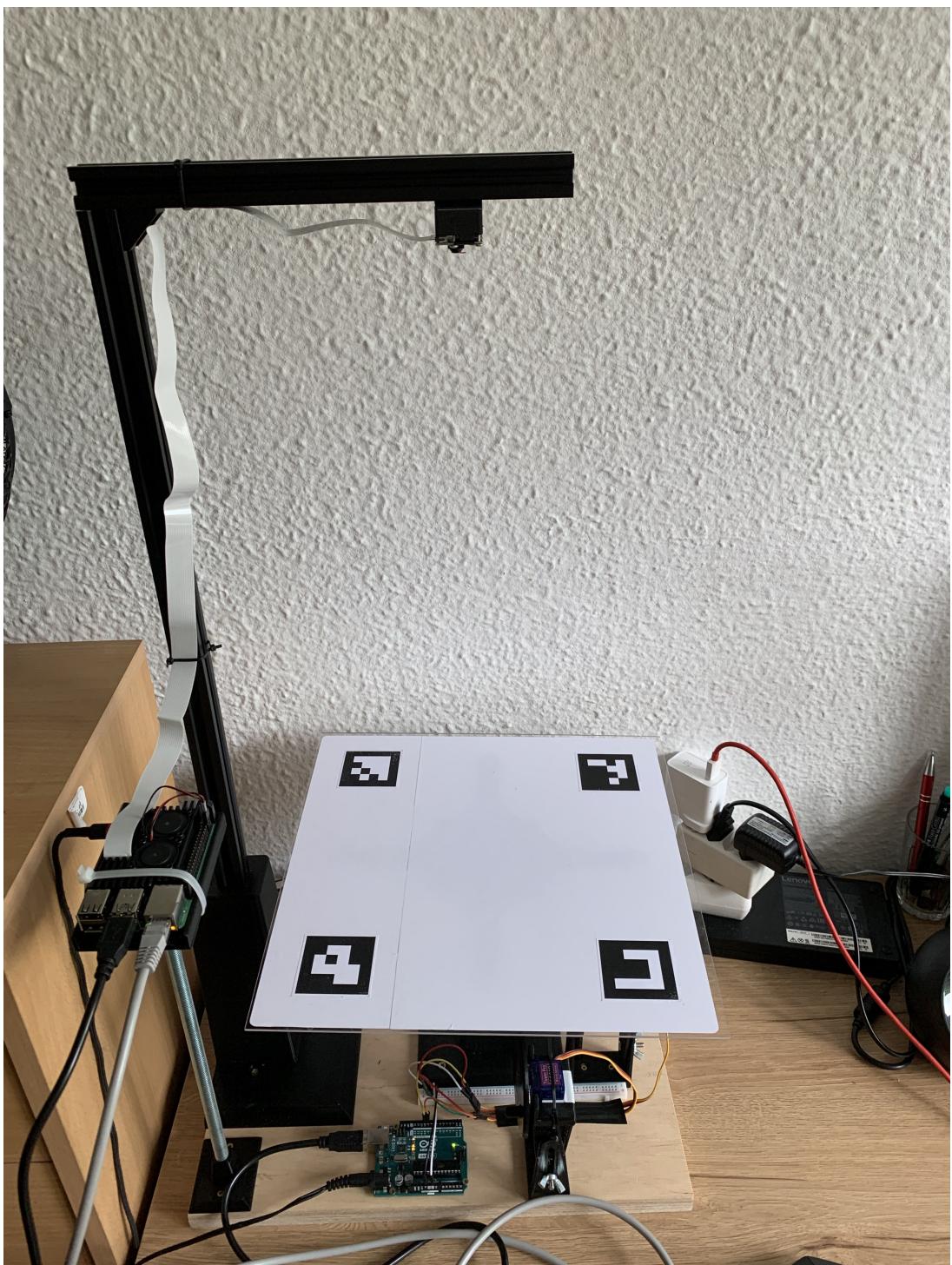


Rysunek 3: Platforma balansująca.

Zdjęcia konstrukcji:



Rysunek 4: Montaż platformy z serwomechanizmami



Rysunek 5: Zdjęcie całej konstrukcji

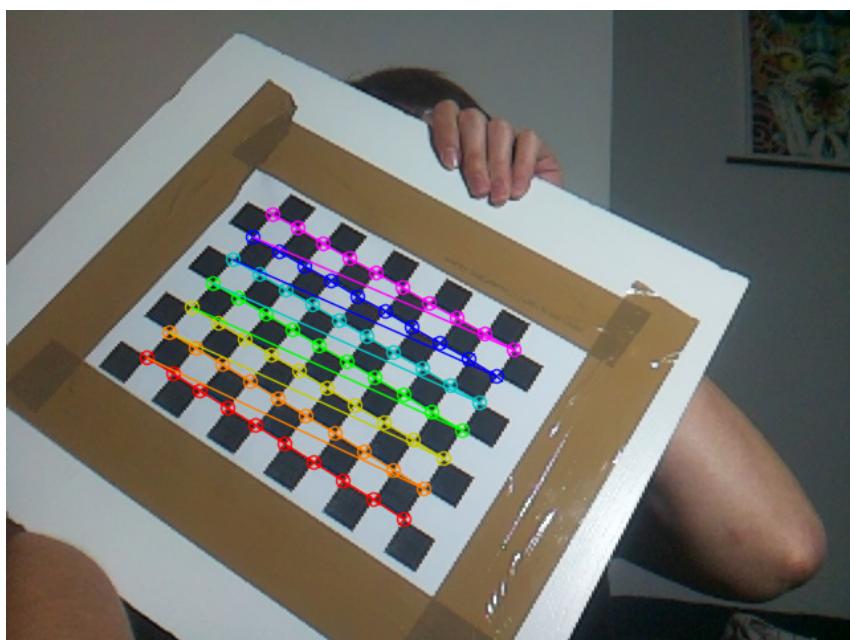
4 Opis działania programu

4.1 Kalibracja kamery

W celu poprawnego wyznaczania pozycji w przestrzeni markerów ArUco, niezbędna była kalibracja kamery i otrzymanie parametrów kamery. Do kalibracji wykorzystane zostały funkcje dostępne w bibliotece OpenCV. Kalibracja odbyła się poprzez analizę zdjęć szachownicy. Każde zdjęcie wejściowe jest analizowane pod kątem wyznaczenia wewnętrznych krawędzi szachownicy. Kalibracja musiała zostać wykonana tylko raz a wszystkie parametry zostały zapisane w pliku .json, który jest odczytywany w głównym programie balansującym piłką.



Rysunek 6: Obraz wejściowy

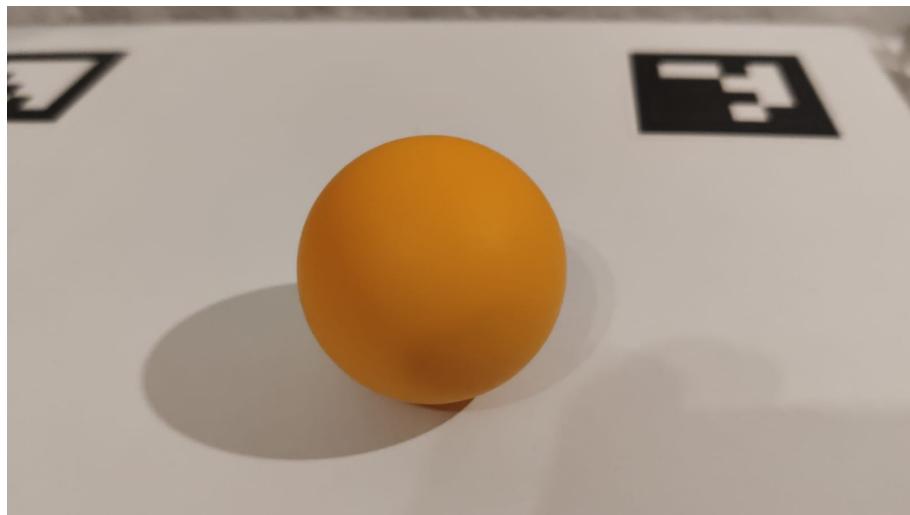


Rysunek 7: Obraz wyjściowy

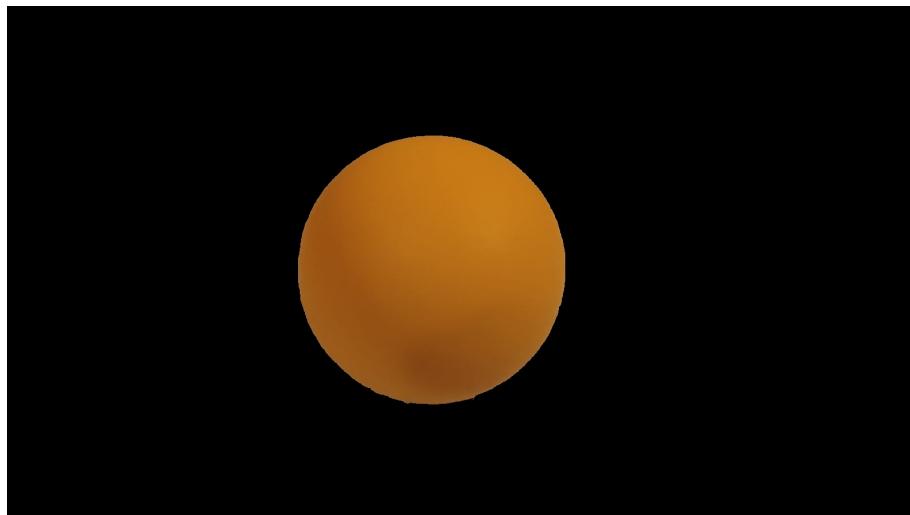
4.2 Wyznaczanie zakresu koloru piłeczki w modelu HSV

W celu poprawnego wyznaczania pozycji piłeczki niezbędne było określenie górnej i dolnej granicy koloru piłeczki. Do wyznaczenia tych granic został wykorzystany skrypt udostępniony w serwisie StackOverflow. Obraz wyjściowy otrzymano dla:

- Granica dolna: ($H = 0$, $S = 150$, $V = 120$)
- Granica górska: ($H = 180$, $S = 255$, $V = 255$)



Rysunek 8: Obraz wejściowy



Rysunek 9: Obraz wyjściowy

4.3 Detekcja markerów

Korzystając z biblioteki OpenCV została zaimplementowana detekcja markerów ArUco oraz estymacja ich pozycji w przestrzeni. Każda klatka jest filtrowana za pomocą filtra bilateralnego [1] w celu wygładzenia detali i wydobycia krawędzi a następnie przetwarzana w celu poszukiwaniu markerów ArUco.

```
def get_ball_dist_from_center(frame, matrixCoeff, distortionCoeff, arucoDetector):
    frame = cv.bilateralFilter(frame, 9, 100, 100)
    gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    markerCorners, markerIDs, rejectedCandidates = arucoDetector.detectMarkers(
        gray)
```

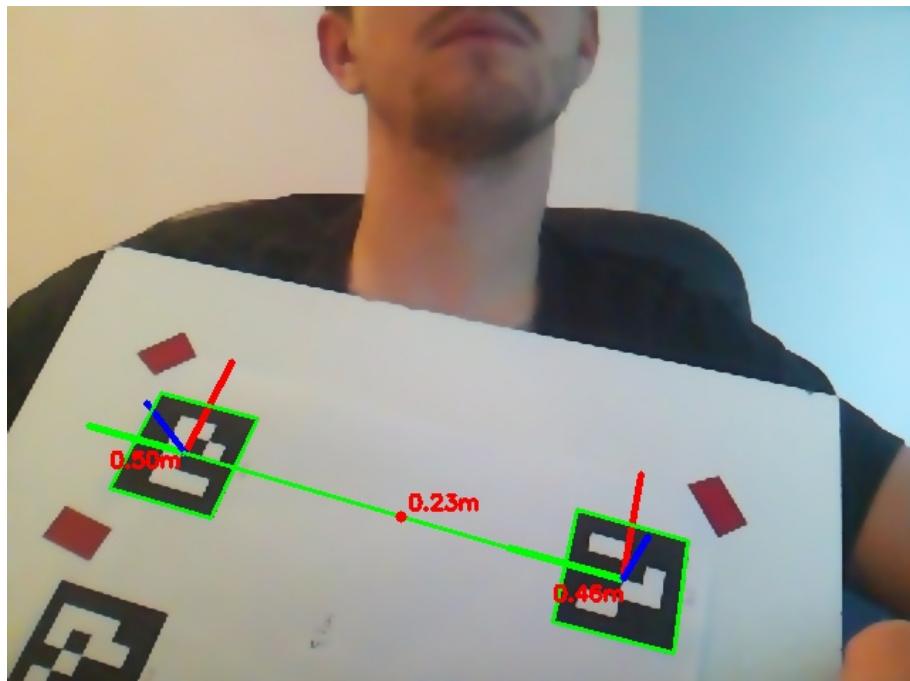
Następnie estymowana i rysowana jest pozycja w przestrzeni każdego wykrytego markera. Do estymacji wykorzystywane są parametry wyznaczone podczas kalibracji kamery.

```
if len(marker_corners) > 0:
    marker_ids = marker_ids.flatten()

    # Estimate pose of the markers in relation to the camera position
    rvec, tvec, _ = cv.aruco.estimatePoseSingleMarkers(marker_corners,
                                                       marker_size_cm, matrix_coeff, distortion_coeff)

    for (marker_corner, _, i) in zip(marker_corners, marker_ids, range(0, marker_ids.size)):
        # Draw frame axes for each marker
        cv.drawFrameAxes(frame, matrix_coeff, distortion_coeff, rvec[i], tvec[i], marker_size_cm)

    if len(marker_corners) > 1:
        for i in range(len(marker_ids)):
            for j in range(i + 1, len(marker_ids)):
                # Calculate the distance between markers
                distance = np.linalg.norm(tvec[i] - tvec[j])
```



Rysunek 10: Przykład detekcji i estymacji pozycji markerów

4.4 Detekcja piłki

W celu detekcji i lokalizacji piłki obraz najpierw jest przetwarzany na model HSV a następnie filtrowany i nakładana jest na niego maska stworzona na podstawie obrazu oraz granicznych wartości koloru w modelu HSV.

```
def detect_ball(frame):
    # Define the lower and upper bounds for the ball color
    lower_color = np.array([0, 150, 190])
    upper_color = np.array([180, 255, 255])

    # Convert the image to HSV color space
    hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)

    # Create a mask for the ball color
    mask = cv.inRange(hsv, lower_color, upper_color)

    # Apply morphological operations to remove noise
    kernel = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5, 5))
    mask = cv.morphologyEx(mask, cv.MORPH_OPEN, kernel)
    mask = cv.morphologyEx(mask, cv.MORPH_CLOSE, kernel)
```

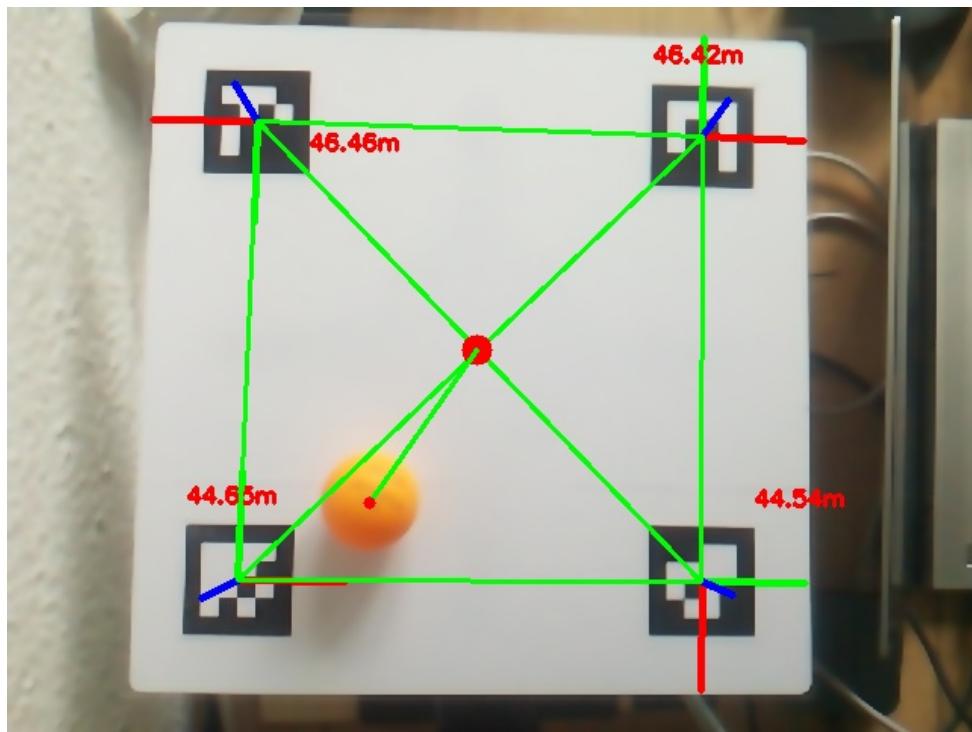
Następnie wyznaczane są wszystkie kontury na obrazie, a największy uznawany jest jako piłka. Centralny punkt konturu jest zwracany jako środek piłki.

```
# Find contours of the ball
contours, _ = cv.findContours(
    mask, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)

# Find the largest contour (assumed to be the ball)
if len(contours) > 0:
    ball_contour = max(contours, key=cv.contourArea)
    (x, y), _ = cv.minEnclosingCircle(ball_contour)
    return (int(x), int(y))

Jeśli nie wykryto piłki na obrazie to zwracany jest null

return None
```



Rysunek 11: Lokalizacja piłki względem środka platformy

4.5 Regulator PID

Została stworzona klasa regulatora przyjmująca odpowiednie nastawy przy wywołaniu oraz metoda tej klasy, której wywołanie powoduje wywołanie samego regulatora.

```
def regulate(self, setpoint_error: int):
    self.distance_error = self.setpoint - setpoint_error

    # Proportional
    self.PID_p = self.k_p * self.distance_error

    # Integral
    if - self.integral_bounds < self.distance_error < self.integral_bounds:
        self.PID_i += self.k_i * self.distance_error
    else:
        self.PID_i = 0

    # Derivative
    self.PID_d = self.k_d * \ ((self.distance_error - self.distance_previous_error)
        ) / self.period)

    self.PID_total = self.PID_p + self.PID_i + self.PID_d
    self.PID_total = self._map_value(self.PID_total, -190, 190, self.
        servo_lower_bound, self.servo_upper_bound)

    if self.PID_total < self.servo_lower_bound:
        self.PID_total = self.servo_lower_bound
    if self.PID_total > self.servo_upper_bound:
        self.PID_total = self.servo_upper_bound

    self.distance_previous_error = self.distance_error

    return int(self.PID_total)
```

Dodatkowo została stworzona metoda pomocnicza mapująca wartość w danym przedziale

```
def _map_value(value, in_min, in_max, out_min, out_max):
    return (value - in_min) * (out_max - out_min) / (in_max - in_min) + out_min
```

4.6 Sterowanie serwomechanizmami

Sterowanie serwomechanizmami odbywa się za pomocą Arduino Uno, które wczytuje nastawy z portu seryjnego a następnie ustawię serwomechanizmy w odpowiedniej pozycji.

```
#include <Servo.h>

Servo myservo_1;
Servo myservo_2;

int servo_1_angle = 60;
int servo_2_angle = 65;

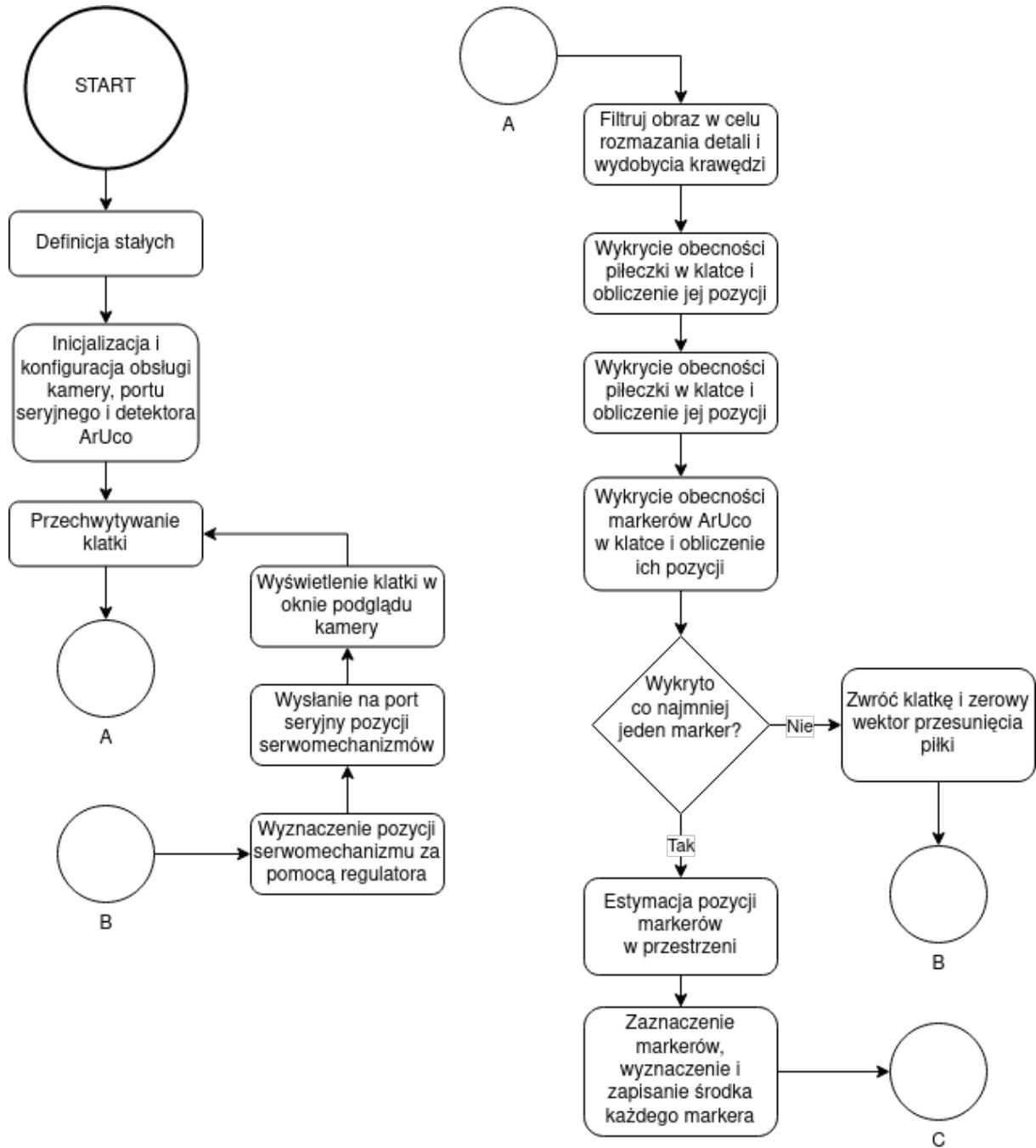
void setup() {
    myservo_1.attach(9);
    myservo_2.attach(10);
    Serial.begin(9600);
}

void loop() {
    if (Serial.available()) {
        String data = Serial.readStringUntil('\n');

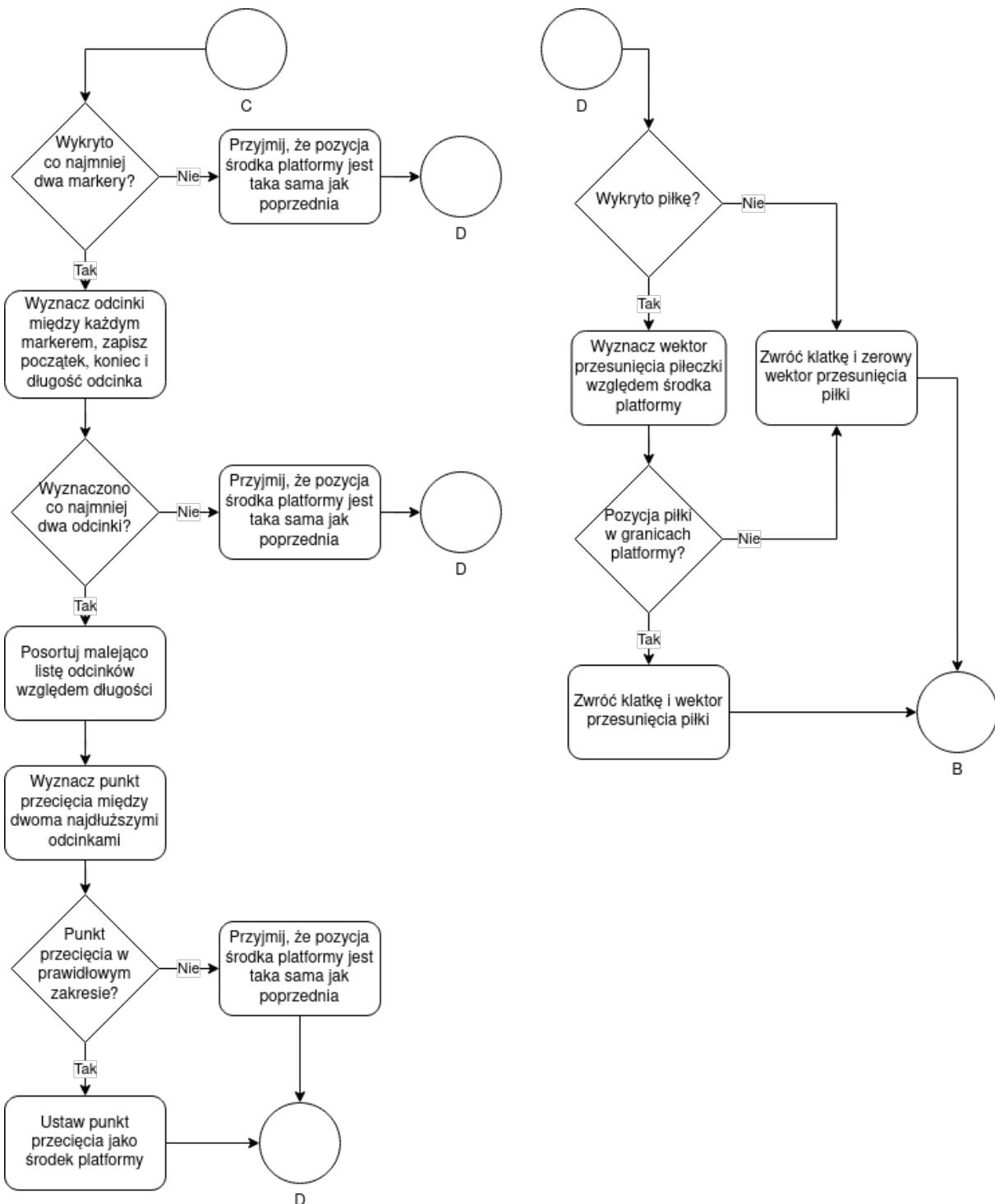
        int separatorIndex = data.indexOf(',');
        if (separatorIndex != -1) {
            String Str1 = data.substring(0, separatorIndex);
            String Str2 = data.substring(separatorIndex + 1);
            Serial.println(data);
            servo_1_angle = Str1.toInt();
            servo_2_angle = Str2.toInt();
        }
        myservo_1.write(servo_1_angle);
        myservo_2.write(servo_2_angle);
    }
}
```

4.7 Główny program

Program balansujący piłkę przetwarza każdą otrzymaną klatkę, wyznacza pozycje markerów ArUco a na podstawie pozycji tych markerów wyznacza środek platformy będący punktem przecięcia przekątnych. Wyznaczana jest też pozycja piłeczki i obliczany jest wektor przemieszczenia piłeczki gdzie punktem odniesienia jest punkt (0, 0) będący środkiem platformy. Przesunięcie jest następnie przekazywane na wejście regulatorów działających niezależnie w swoich osiach, które dobierając odpowiednio pozycje serwomechanizmów, przesuwają piłkę do punktu (0, 0) i utrzymują ją w tej pozycji. Poniżej zamieszczony został diagram przepływu programu



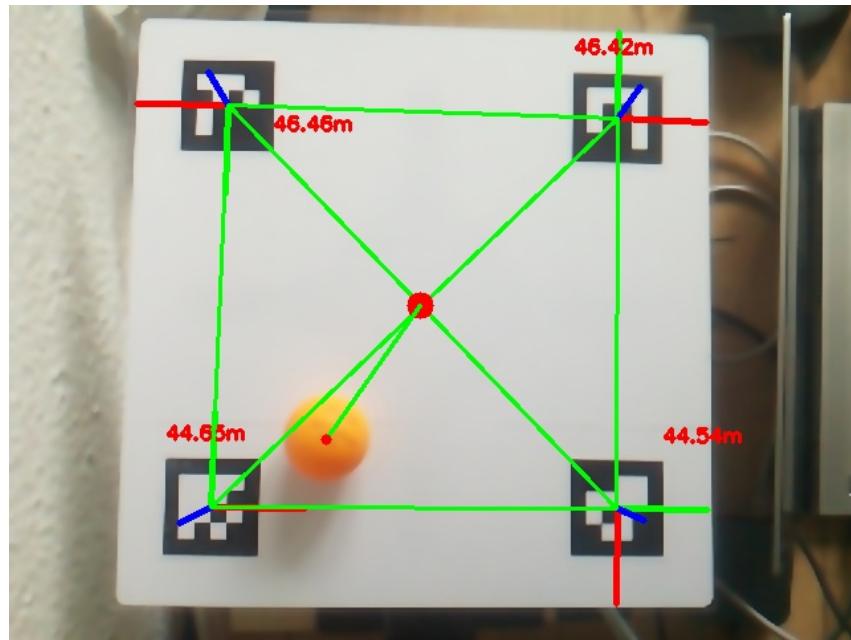
Rysunek 12: Diagram przepływu



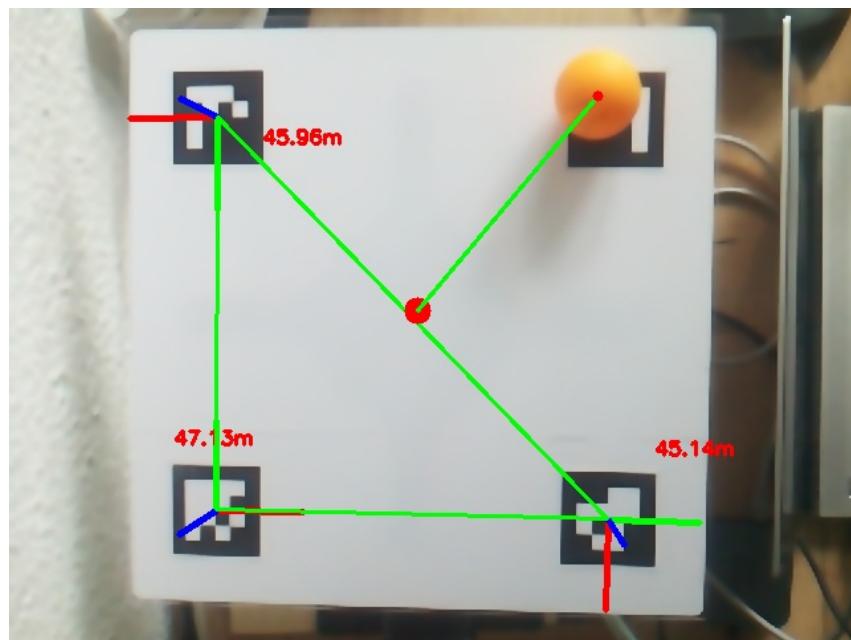
Rysunek 13: Diagram przepływu

Główna pętla programu

```
while True:  
    frame, result = get_ball_dist_from_center(camera.capture_array(), mtx,  
                                              dist, arucoDetector)  
  
    servo_value_y = PID_y.regulate(-result[0])  
    servo_value_x = PID_x.regulate(result[1])  
    serial_port.write(b"%d,%d\n" % (servo_value_y, servo_value_x))  
  
    time.sleep(period)
```

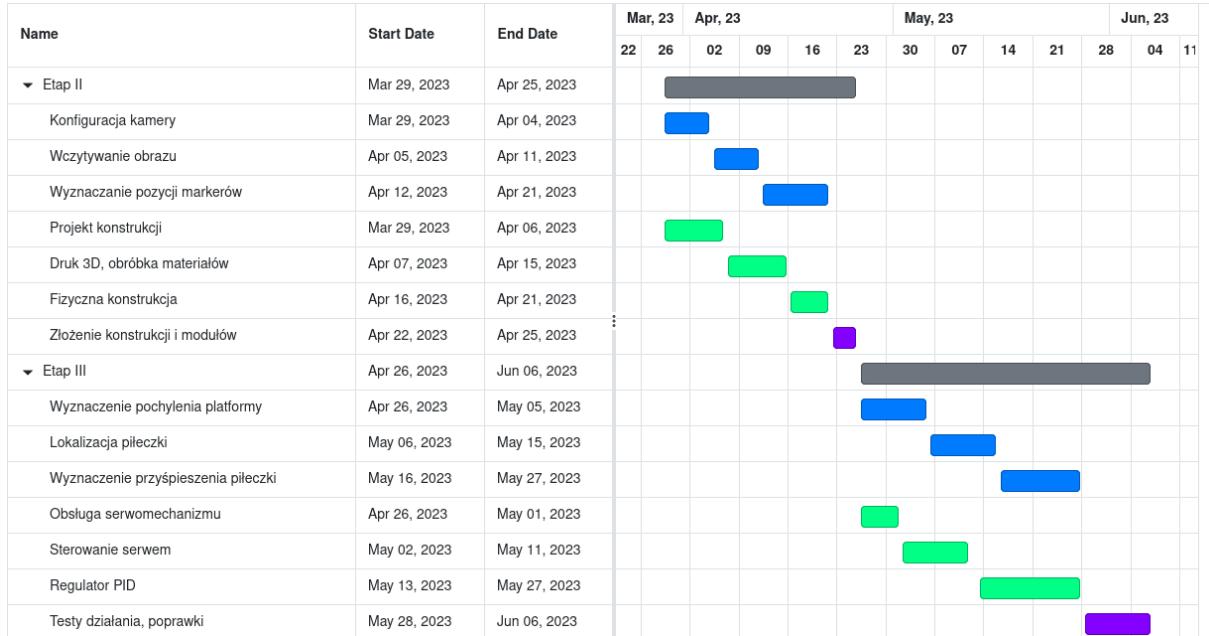


Rysunek 14: Lokalizacja piłki względem środka platformy



Rysunek 15: Poprzedni środek platformy jest zapamiętywany jeśli nie wszystkie markery są widoczne

5 Harmonogram pracy



Rysunek 16: Diagram Gantta uzyskany za pomocą serwisu Online Gantt.

Legenda: Kolor niebieski: Michał Trela, Kolor zielony: Jan Masłowski, Kolor fioletowy: cele wspólne

5.1 Podział pracy

Michał Trela	Jan Masłowski
Konfiguracja interfejsu kamery	Stworzenie projektu konstrukcji mechanicznej
Wczytywanie obrazu z kamery	Druk 3D elementów, obróbka materiałów
Wyznaczanie pozycji markerów Aruco na podstawie wczytanego obrazu	Wykonanie fizycznej konstrukcji
Złożenie konstrukcji i modułów elektronicznych w całość	

Tabela 4: Podział pracy – Etap II

Michał Trela	Jan Masłowski
Wyznaczenie pochylenia platformy	Obsługa serwomechanizmu
Lokalizacja piłeczki na platformie	Sterowanie serwomechanizmem z różnymi prędkościami
Wyznaczanie wartości i kierunku przyśpieszenia piłeczki	Implementacja regulatora PID
Testy działania oraz końcowe poprawki	

Tabela 5: Podział pracy – Etap III

5.2 Kamienie milowe

- Obsługa kamery - Transmisja obrazu w czasie rzeczywistym oraz wczytywanie obrazu jako tablica do programu [3] [4] [5]
- Stworzenie fizycznej konstrukcji - Wydrukowanie i złożenie elementów w całość

- Obsługa serwomechanizmu - Możliwość sterowania serwomechanizmem z różnymi prędkościami
- Lokalizacja piłki - Wczytany obraz jest analizowany, określana jest dokładna pozycja piłki oraz pozycja markerów [2]
- Implementacja regulatora PID - Implementacja regulatora w oprogramowaniu, wyznaczone wszystkie nastawy [6] [7]
- Balansowanie piłką - Robot balansuje piłką na platformie bez udziału człowieka

6 Dokumentacja projektu

Kod źródłowy oraz dokumentacja zostały zamieszczone w serwisie GitHub:

<https://github.com/Mmichal1/PID-Ball-Balancing-Platform.git>

Prezentacja konstrukcji oraz działania platformy została udostępniona w serwisie YouTube

https://youtu.be/hny_5dCIR1Q

Literatura

- [1] A gentle introduction to bilateral filtering and its applications. http://people.csail.mit.edu/sparis/bf_course/#intro.
- [2] Opencv documentation. https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html.
- [3] G. Bradski, A. Kaehler, i in. Opencv. *Dr. Dobb's journal of software tools*, 3(2), 2000.
- [4] F. Jalled, I. Voronkov. Object detection using image processing. *CoRR*, abs/1611.07791, 2016.
- [5] M. A. Pagnutti, R. E. Ryan, G. J. C. V, M. J. Gold, R. Harlan, E. Leggett, J. F. Pagnutti. Laying the foundation to use Raspberry Pi 3 V2 camera module imagery for scientific and engineering purposes. *Journal of Electronic Imaging*, 26(1):013014, 2017.
- [6] E. Sariyildiz, H. Yu, K. Ohnishi. A practical tuning method for the robust pid controller with velocity feed-back. *Machines*, 3(3):208–222, 2015.
- [7] K. Yaovaja. Ball balancing on a stewart platform using fuzzy supervisory pid visual servo control. *2018 5th International Conference on Advanced Informatics: Concept Theory and Applications (ICAICTA)*, strony 170–175, 2018.