KAUNO TECHNOLOGIJOS UNIVERSITETAS
ELEKTROS IR ELEKTRONIKOS FAKULTETAS

Procesų valdymas realiuoju laiku (T125B407)

Kursinio darbo ataskaita

„Objektų sekimo realizavimas naudojant kompiuterinę regą“

Darbą parengė:
Viktoras Januška E VS - 9
Darbą priėmė:
Lekt. Kęstas Rimkus

Kaunas, 2022

Įvadas

Šiuolaikiniame pasaulyje vis dažniau yra naudojami sekimo algoritmai. Ginkluotėje naudojamos autonominės raketos, kurios gali numatyti ir sekti taikinio buvimo vietą. Kai kuriose šalyse naudojamos kamerų sistemos žmonių sekimui ir apsaugai. Vykdomos naujos virtualios realybės kompiuterinių žadimų technologijos, kad būtų galima nustatyti žmogaus padėtį trimatėje erdvėje. Taip pat, objekto sekimo algoritmus galima panaudoti kameros pozicionavimui ervėje. Tokiu atveju nereikalingas papildomas žmogus – operatorius. Šiame darbe ir bus projektuojamas toks robotas, kuris aptinka žmogų ar raudoną kamuoliuką ir stengiasi centruoti save taip, kad sekamas objektas būtų kadro viduryje.

Darbo tikslas

Pagaminti robotą, kuris naudodamas kompiuterinės regos algoritmus ir PID reguliatorių keistų kameros polinkį ir pokrypį taip, kad sekamo objekto centras būtų kadro viduryje, tuo tarpu prie roboto būtų galima pritvirtinti kamerą ar telefoną, kuris filmuotų ar darytų nuotraukas. Panaudoti 2 sekimo algoritmo alternatyvas:

1) Pasirinktos spalvos objekto sekimas;

2) Žmogaus veido atpažinimas ir sekimas;

Roboto valdymui bus panaudotas mikrovaldiklis programojamas C++ kalba, kuris per Serial komunikaciją gaus iš kompiuterio duomenis apie pasisukimo kampą. Kompiuteryje vaizdo apdorocijmui bus panaudota Python kalba.

Darbo uždaviniai

1. Suprojektuojamas ir konstruiojamas roboto 3D modelis;

2. Paruošiama kamera;

3.Sukuriamas sekimo algoritmas tam tikros spalvos objektui

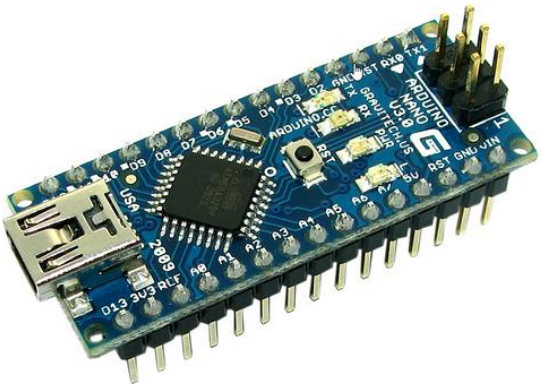
4.Sukuriamas sekimo algoritmas žmogaus veidui.

5. Sukuriamas PID reguliatorius, kuris centruoja objekto vaizdą kadro centre.

Darbe naudotos detalės

1) **“Arduino NANO”**

Panaudota Arduino UNO plokštė. Tai - “Geekcreit” Arduino plokštės variantas gautas iš Kinijos. Jo pagrindas yra ATmega328P mikrovaldiklis. Dėl patogaus naudojimo, prieinamos kainos ir nedidelių matmenų buvo pasirinkta šis mikrovaldiklis.



1 pav. Arduino nano

2) **2 x "MG995" Servo pavaros**

Panaudoti du MG995 Servo motorai. Pasirinkti didesni motorai dėl jėgos reikalingos pakelti telefoną ar kamerą (apie 9kg); Vienas naudojamas kameros pokrypiui, kitas polinkiui reguliuoti.

3) **„Sony Playstation 3 Eye" kamera**

Pasirinkta ganėtinai sena kamera, kuri gali filmuoti iki 120 kadrų per sekundę naudodama rezoliuciją: 320 x 240. Dėl gana aukšto kadrų dažnio, kamera užtikrina reguliatoriaus greitaveiką.

2 pav. PS3 Eye kamera

4) **Nuosavas kompiuteris:**

Skirtas prijungti kamerą, objektų atpažinimui ir Serial komunikacijai su mikrovaldikliu.

5) **Maitinimo šaltinis**

Panaudotas 5V maitinimo šaltinis tiekiantis iki 3A srovę, tinkantis servo variklių maitinimui.

6) **Roboto korpusas**

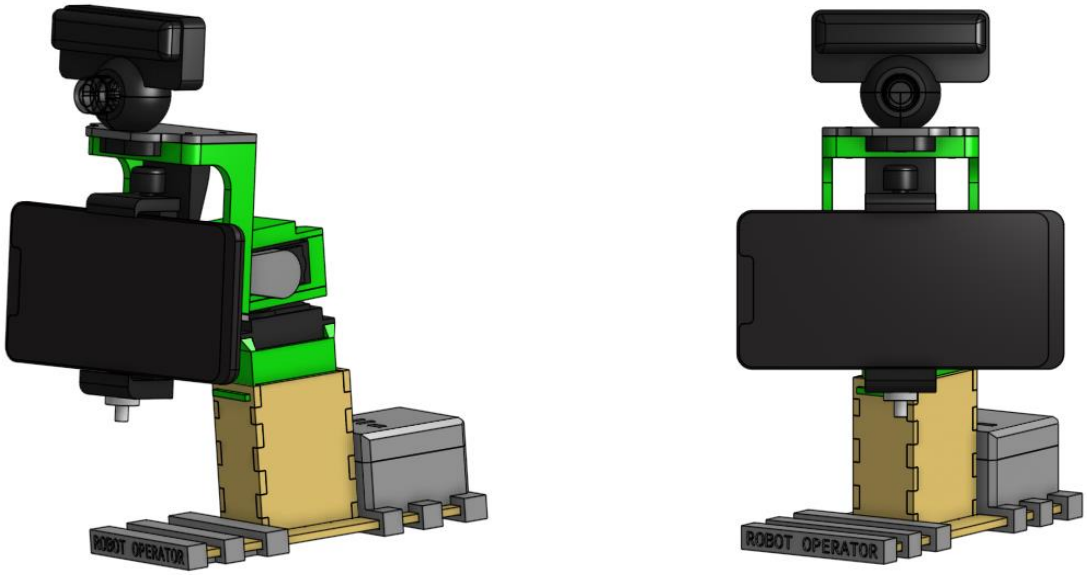
7) **Papildomas lęšis:**

Naudojamas papildomas lęšis kamerai su trumpesniu židinio nuotoliu. Tai leidžia matyti platesnį vaizdą ir duoda mažesnę tikimybę objektui pradingti iš kameros matymo lauko.

3 pav. pakeičiamas lęšis

3D modelio kūrimas ir spausdinimas

„Onshpe“ grafinio dizaino programa nubraižytas roboto modelis su tiksliais išmatavimais, atitinkamos dalys eksportuotos stl ir dxf formatu, kad būtų galima jas spausdinti 3d spausdintuvu arba išpjauti staklėse. Galiausiai detalės pagamintos ir robotas sukonstruotas.



4 pav. Roboto 3D modelis

Kameros paruošimas

Kameros vaizdui nuskaityti ir kompiuterinei regai vykdyti bus panaudojama „OpenCV“ biblioteka naudojantis „Python“ programavimo aplinkoje.

Standartinis „PS3 Eye” kameros lęšis pakeičiamas į trumpesnį židinio nuotolį turinį lęšį. Dėl to, gauname netiesinį kameros vaizdą: artimi objektai atrodo deformuoti ir nenatūralūs – yra iškraipomi. Kadro taškai gali būti iškraipomi pagal radialinį arba tangentinį iškraipymą.

Radialinio iškraipymo metu tiesės gali atrodyti kaip kreivės. Ir šis iškraipmas darosi stipresnis kuo taškai yra toliau nuo kadro centro. Jis apibūdinamas tokiomis formulėmis:

$$x_{dist} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{dist} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$\text{Čia: } r - \text{ nuotolis nuo iškreipto taško ir iškraipymo centro,}$$
$$k_1, k_2, k_3 - \text{ radialinio iškraipymo koeficientai}$$

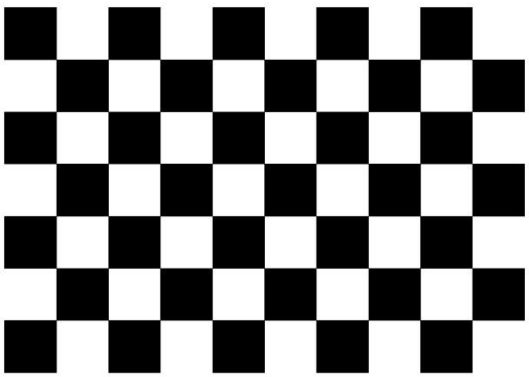
Tangentinis iškraipymas atsiranda tada, kai lęšis nėra pilnai lygiagretus vaizdavimo plokštumai (kameros jutikliui). Jis apibūdinamas tokiomis formulėmis:

$$x_{dist} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$
$$y_{dist} = y + [p_1(r^2 + 2y_2) + 2p_2 xy]$$

$$\text{Čia: } p_1, p_2 - \text{tangentinio iškraipymo koeficientai}$$

Kadangi iš kadro norime išgauti kuo tikslesnę kameros orientaciją objekto atžvilgiu, šie netiesiškumai turi būti panaikinami. Tam panaudojama vaizdo ištiesinimo transformacija.

Pirmiausia, turime surasti kameros matricą ir iškraipymo koeficientus. Panaudojamas šachmatų lentos metodas. Šio metodo metu panaudojame atspausdintą baltų ir juodų kvadratų, kurių matmenis žinome, šabloną.



5 pav. Šachmatinės lentos šablonas

Naudojama kamera fotografuoja šį šabloną taip, kad jis visas tilptų į kadrą. Reikia fotografuoti keičiant šablono orientaciją ir atstumą iki kameros, kad būtų tiksliai matoma, kur atsiranda iškraipymai.

Kiekvienam nufotografuotam vaizdui panaudojama funkcija „OpenCV“ bibliotekoje “cv2.findChessboardCorners„ , kad būtų surandami kvadratų susilietimo kampai, prieš tai vaizdas paverčiamas į nespalvotą formatą.

```
img      = cv2.imread(fname)
gray     = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
```

6 pav. Vaizdo nuskaitymo funkcija

```
ret, corners = cv2.findChessboardCorners(gray, (nCols,nRows),None)
```

7 pav. Kvadratų kampų taškų suradimo funkcija

Šie kampai panaudojami funkcijoje „cv2.calibrateCamera“, iš kurios gauname kameros matricą ir iškraipymo koeficientus.

```
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1],None,None)
```

8 pav. Kameros ir iškraipymo taškų suradimo funkcija

Iškraipymo koeficientai apibūdinami tokiu vektorium:

$$D = [k_1, k_2, p_1, p_2, k_3]$$

Kameros matrica abibūdinama taip:

$$C = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Čia: $f_x, f_y - židinio\ nuotolis\ x\ ir\ y\ atžvilgiu, c_x, c_y - lęšio\ optinis\ centras$

Šie gauti koeficientai įrašomi prieš sekimo kodo pagrindinį ciklą

```
self.mtx =  np.matrix([[435.39791874,   0.0,         339.93017819],
                       [  0.0,         433.39637602,     252.35640574],
                       [  0.0,           0.0,           1.0        ]])

self.dist =  np.matrix([[-0.31499994,  0.12603356,  0.00062101,  0.00160527, -0.0262532 ]])
```

9 pav. Gauti kameros koeficientai

Suradę juos, galime vykdyti vaizdo ištiesinimo transformaciją. Prieš vykdant pagrindinį ciklą, randama optimali kameros matrica.

```
self.newcameramtx, self.roi=cv2.getOptimalNewCameraMatrix(self.mtx,self.dist,(self.w,self.h),1,(self.w,self.h))
```

10 pav. Optimalios kameros matricos radimo funkcija

Po kiekvieno kadro nuskaitymo vykdomas ištiesinimas.

```
def undistort(self, img):
    dst = cv2.undistort(img, self.mtx, self.dist, None, self.newcameramtx)
    return dst
```

11 pav. ištiesinimo funkcija

Pasirinktos spalvos objekto sekimas

Pirmiausia vykdomas vaizdo nuskaitymas ir vaizdo kalibravimas. Tada pakeičiamas spalvos formatas ir surandami norimų spalvų intervalo taškai kadre – kuriamas užmaskavimo sluoksnis. Šiam sluoksniui atliekama vaizdo erozija, taškai dalinai išardomi. Tada jie išplečiami. Norint matyti tik išfiltruotą vaizdą atliekame “IR” operaciją tarp užmaskavimo sluoksnio ir pradinio nuskaityto vaizdo. Toliau atliekame kontūrų paiešką ir randame išfiltruotos spalvos kontūrus.

```
success, image = self.cap.read()
undistorted = self.undistort(image)
hsv_frame = cv2.cvtColor(undistorted, cv2.COLOR_BGR2HSV)
mask = cv2.inRange(hsv_frame, (10,130,130), (40,255,255))
mask = cv2.erode(mask, None, iterations=2)
mask = cv2.dilate(mask, None, iterations=2)
result = cv2.bitwise_and(image, image, mask=mask)

cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
center = None
```

12 pav. Maskavimo sluoksnio kūrimas ir šio sluoksnio kontūrų radimas

Surandame didžiausio kontūro centroido koordinates, kurios bus panaudotos pasisukimo kampui rasti ir jo spindulį.

```
c = max(cnts, key=cv2.contourArea)
((x, y), radius) = cv2.minEnclosingCircle(c)
M = cv2.moments(c)
center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
```

13 pav. Didžiausio kontūro centroido radimas

Sekimas vykdomas tik jeigu centroido spindulys yra didesnis negu 3 pikseliai. Ant centroido nupiešiamas apskritimas.

Išskaičiuojami aptikto objekto kampai nuo centrinės kameros ašies ties X ir Y ašimis. Tam panauodojama „map“ funkcija, kuri norimą reikšmę pirmame intervale, pakeičia kita reikšmę antrame intervale.

```
self.angle_X = self.fov/2 - self.map(x, 0, self.w, 0, self.fov)
self.angle_Y = (self.fov*self.aspect_ratio/2 - self.map(y, 0, self.h, 0, self.fov*self.aspect_ratio))*-1
```

14 pav. Polinkio ir pokrypio kampų radimas

Žmogaus veido sekimas

Žmogaus veido atpažinimui naudojama “MediaPipe” biblioteka. Galima naudoti ir “OpenCV” atpažinimo algortimą, tačiau “MediaPipe” pasižymi tuom, kad atpažinimas atliekamas labai greitai (iki 100 kadrų per sekundę) ir veidas gali būti atpažintas iš visų pusių arba įvairiai pakreipus.

Pirmiausia, sukuriamas veido atpažinimo objektas.

```
self.mpFaceDetection = mp.solutions.face_detection
self.faceDetection = self.mpFaceDetection.FaceDetection()
```

15 pav. Veido atpažinimo objekto sukūrimas

Tada, nuskaitomas kameros kadras ir vaizdas ištiesinamas, taip pat reikia pakeisti spalvų formatą iš “BGR” į “RGB”. Galiausiai vykdomas veido atpažinimas.

```
success, image = self.cap.read()
undistorted = self.undistort(image)
imgRGB = cv2.cvtColor(undistorted, cv2.COLOR_BGR2RGB)
results = self.faceDetection.process(imgRGB)
```

16 pav. Nuskaitytas kadras panaudojamas veido atpažinimo funkcijoje

Atlikus veidų atpažinimą, sekimą naudojame tada tik kai randamas vienas veidas. Tada gauname aptikto veido apribojamo stačiakampio koordinates. Tada brėžiame liniją nuo šio stačiakampio centro iki kadro centro.

```
bbox = results.detections[0].location_data.relative_bounding_box
pixel_x = bbox.xmin + bbox.width/2
pixel_y = bbox.ymin + bbox.height/2
cv2.line(undistorted, (int(pixel_x*self.w),int(pixel_y*self.h)), (int(self.w/2), int(self.h/2)), (255, 0, 0), 3)
```

17 pav. Randamas veidą apribojantis stačiakampis

Apskaičiuojamas reikalingi kameros pasisukimo kampai ties X ir Y ašimi naudojant apribojamo stačiakampio koordinates.

```
self.angle_X = self.fov/2 - self.map(pixel_x*self.w, 0, self.w, 0, self.fov)
self.angle_Y = (self.fov*self.aspect_ratio/2 - self.map(pixel_y*self.h, 0, self.h, 0, self.fov*self.aspect_ratio))*-1
```

18 pav. Randami veido orientacijos kampai nuo kameros centrinės ašies

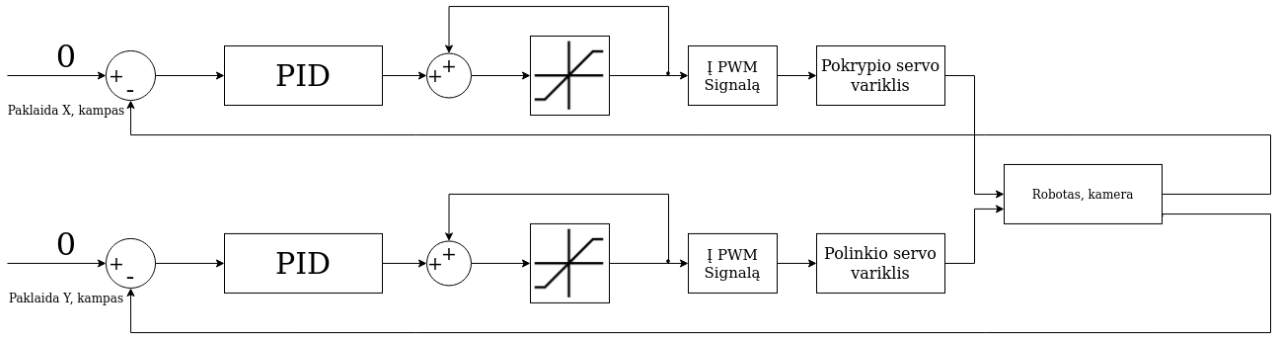
Servo variklių valdymas

Servo varikliai valdomi tiekiant PWM (Impulso pločio moduliacija) signalą į signalo jungtį. PWM signalas tai pastoviai tiekiamas įsijungiantis/išsijungiantis elektrinis potencialas. Padidinant arba sumažinant įjungto signalo laiką galima valdyti įvairius komponentus, pvz: LED lemputės ryškumą. Įjungto potencialo laikas vadinamas darbinguoju ciklu(angl. Duty

Cycle). Šio roboto atveju, signalas tiekiamas servo varikliams, todėl turi būti siunčiamas 50Hz dažniu. MG995 servo variklių atveju, keičiant teigiamos įtampos laiko periodą nuo 0,6 milisekundės iki 2,2 milisekundžių servo variklis atitinkamai juda nuo 0 laipsnių pozicijos iki 180 laipsnių pozicijos. Tam galima panaudoti "Servo" biblioteką Arduino programavimo aplinkoje.

Bendras sistemos valdymo supratimas

Šiame darbe roboto valdymą (orientacijos reguliavimą) atlieka mikrovaldiklis "Atmega328P". Valdymo įėjimas visada pastovus, tai yra 0. Nuosavas kompiuteris naudodamas kamerą aptinka objektą ir išskaičiuoja pokrypio ir polinkio paklaidas. Šios paklaidos per "Serial" komunikaciją yra siunčiamos valdikliui. Valdiklis naudoja 2 atskirus diskretinius PID reguliatorius šiai paklaidai sumažinti (Vieną pokrypiui, kitą polinkiui). Gautoji reikšmė pridedama prie žinomos variklių pasisukimo reikšmės. Jeigu šių reguliatorių išėjimas didesnis negu leistinasis, jis yra sumažinamas iki leistinojo. Šią reikšmę tada paverčiame į PWM signalą ir siunčiame Servo varikliams. Juose yra atskira uždara valdymo sistema, kuri palaiko užduoto PWM signalo atitinkamą pasisukimo kampą.



19 Valdomos sistemos modelis

Diskretinės valdymo sistemos realizavimas mikrovaldiklyje

Mikrovaldiklis programuojamas Arduino programavimo aplinkoje. Trumpai panagrinėsime šio valdiklio kodą.

Prieš pradedant pagrindinį ciklą mikrovaldiklyje, aprašomi kintamieji PID reguliatorių veikimui. Tai PID konstantos, pradinė Servo variklių padėtis, paklaida, paklaidos pakitimas, paklaidos plotas ir paklaidos išvestinė. Kad galėtume išskaičiuoti PID reguliatoriaus laiko pokytį, randama pradinė laiko reikšmė.

```
double kp_X = 0.12;
double kd_X = 800;
double ki_X = 0.000001;
double kp_Y = 0.08;
double kd_Y = 1200;
double ki_Y = 0.00000001;
```

20 pav. PID reguliatoriaus konstantų užrašymas

```
int X_servo = 900;
int Y_servo = 600;
float X_error = 0;
float X_error_old = 0;
float X_error_change = 0;
float X_error_slope = 0;
float X_error_area = 0;

float Y_error = 0;
float Y_error_old = 0;
float Y_error_change = 0;
float Y_error_slope = 0;
float Y_error_area = 0;

int timeOld = millis();
int timeNew;
int dT;
int lastServoMove;
```

21 pav. Tarpinių kintamųjų užrašymas

Toliau paskiriami diskretūs valdiklio išėjimai, prie kurių prijungtas servo pavarų signalo įėjimas. Šį signalą keičiame nuo 1 iki 2 milisekundžių atitinkamai keisdami pavaros kampą. Taip pat sukuriama serial komunikacija su kompiuteriu, kad būtų galima nuskaityti paklaidos vertes.

```
void setup() {
  YServo.attach(6);
  XServo.attach(5);
  Serial.begin(115200);
  Serial.setTimeout(1);
  XServo.writeMicroseconds(map(X_servo, 1800, 0, 1000, 2000));
  YServo.writeMicroseconds(map(Y_servo, 0, 1800, 1000, 2000));
  lastServoMove = millis();
  String X_string= String(X_servo);     // empty string
  String Y_string= String(Y_servo);     // empty string
  Serial.println(dataOut);
}
```

22 pav. "setup" funkcija

Pagrindinio ciklo pradžioje tikriname, ar serijiniame buferyje gavome duomenų, mūsų atveju daugiau, negu 8 baitus. Jeigu daugiau, nuskaitoma raidžių eilutė.

```
void loop() {
    if(Serial.available() > 8){
      data = Serial.readString();
      X_error = parseDataX(data);
      Y_error = parseDataY(data);
```

23 pav. Duomenų iš kompiuterio nuskaitymas

Iš pradinės simbolių eilutės yra atitinkamai išgaunamos polinkio ir pokrypio paklaidų simbolių eilutės. Tada jos paverčiamos į tinkamą duomenų formatą – sveikąjį skaičių.

```
int parseDataX(String data){
    //Serial.print(data);
    //Serial.print('\n');
    data.remove(data.indexOf("Y"));
    data.remove(data.indexOf("X"), 1);
    //Serial.print(data.toInt());
    //Serial.print('\n');
    return data.toInt();
}

int parseDataY(String data){
    data.remove(0,data.indexOf("Y") + 1);
    return data.toInt();
}
```

24 pav. Sveikųjų skaičių gavimas iš simbolių eilučių

Gavę naujas paklaidos reikšmes, išskaičiuojame laiko pokytį ir PID reguliatoriaus dedamąsias, kurios yra pridedamos prie dabartinės servo pavaros kampo reikšmės.

```
timeOld = timeNew;
timeNew = millis();
dT = timeNew -timeOld;

X_error_old = X_error;
Y_error_old = Y_error;
X_error_change = X_error - X_error_old;
Y_error_change = Y_error - Y_error_old;
X_error_slope = X_error_change/dT;
Y_error_slope = Y_error_change/dT;
X_error_area = X_error_area+X_error*dT;
Y_error_area = Y_error_area+Y_error*dT;

X_servo = X_servo -( kp_X*X_error + kd_X*X_error_slope + ki_X*X_error_area);
Y_servo = Y_servo +( kp_Y*Y_error + kd_Y*Y_error_slope + ki_Y*Y_error_area);
```

25 pav. PID reguliatoriaus dedamųjų skaičiavimas

Galiausiai atliekame pavarų kampų apribojimus ir išsiunčiame signalą joms. Taip pat, po kiekvieno ciklo negalime keisti servo varikliams tiekiamo PWM signalo, tad po kiekvieno pasisukimo nuskaitomas laikas, kada buvo įvykdytas pasisukimas ir leidžiama vėl keisti signalą tik kai praėjo daugiau, negu viena milisekundė.

```
if(millis() - lastServoMove > 1){
  X_servo = maximumX(X_servo);
  Y_servo = maximumY(Y_servo);
  XServo.writeMicroseconds(map(X_servo, 1800, 0, 1000, 2000));
  YServo.writeMicroseconds(map(Y_servo, 0, 1800, 1000, 2000));
  lastServoMove = millis();
}
```

26 pav. PWM signalo siuntimas varikliams

```
int maximumX(int x){
  if(x >=1800){
    return 1800;}
  else if(x <=0){
    return 0;}
  else return x;
}
int maximumY(int y){
  if(y >=1000){
    return 1000;}
  else if(y <=0){
    return 0;}
  else return y;
}
```

27 pav. Ribojimo funkcija

Serial komunikacija

Duomenys pasirinkti „X laipsnių*10 Y laipsnių*10“ formatu. Siunčiamas simbolių eilutės (String) duomenų paketas. Ir siunčiami tik sveikieji skaičiai, nes su plaukiojančio kablelio skaičiais dažnai duomenys netinkamai nuskaitomi.

Apačioje pateikta, kaip sukuriamas Serial komunikacijos objektas ir panaudojamas duomenim išsiųsti.

```
self.serialas = serial.Serial(self.usb_port, baudrate = 115200, timeout = 1)
```

28 pav. Komunikacijos objekto sukūrimas

```
sk = 'X{:d}Y{:d}\n'.format(int(self.angle_X*10), int(self.angle_Y*10))
self.SendError.emit(self.detection_time, self.angle_X, self.angle_Y)
if self.serialas != False:
    self.serialas.write(sk.encode())
```

29 pav. Paklaidos formatavimas į simbolių eilutę ir išsiuntimas per komunikaciją

Išvados

Sukurtas robotas sekioja norimus objektus naudodamas PID reguliatorių. Robotas sekioja gana nemažu greičiu ir statinė paklaida visada artėja į nulį, nebent objektas pradingsta iš kadro. Kuriant šį projektą susipažinta su kameros kalibravimo procesu, išmokta suprogramuoti diskretinį PID reguliatorių ir panaudoti paskirtai užduočiai. Išmokta atlikti spalvų filtravimą ir surasti atitinkamos spalvos objekto centrą vaizde. Taip pat pagilintos žinios Python ir C++ programavimo kalboje.

Informacijos šaltinių sąrašas

- https://learnopencv.com/camera-calibration-using-opencv/
- https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html
- https://google.github.io/mediapipe/solutions/face_detection.html
- https://www.teachmemicro.com/arduino-pid-control-tutorial/