Wspomaganie mocowania naczepy za pomocą znaczników Aruco i Raspberry PI

Przemysław Dragańczuk, Marcin Serafin, Aksel Stankiewicz

33INF–SSI/A

1 Cel Projektu

Celem projektu jest napisanie programu, który wykrywałby, do której naczepy podłączana jest ciężarówka, a także wspomaganie procesu mocowania naczepy przekazując informacje odnośnie pozycji tej naczepy w stosunku do ciężarówki.

Program powinien wykrywać kod Aruco, znajdujący się na naczepie. Następnie program powinien wysyłać numer ciężarówki, odczytany z kodu, a także pozycję w poziomie naczepy. Odczytane dane powinny być następnie wysyłane przez port szeregowy.

2 Przebieg projektu

2.1 Zainstalowanie OpenCV

Pierwszym krokiem było zainstalowanie biblioteki OpenCV. Użyto do tego poniższego skryptu

```
#!/bin/sh
# Install required packages
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install build-essential cmake pkg-config libjpeg-dev libtiff5-dev
    libjasper-dev libpng12-dev libavcodec-dev libavformat-dev libswscale-dev libv4l-
    dev libxvidcore-dev libx264-dev libgtk2.0-dev libgtk-3-dev libatlas-base-dev
    gfortran python2.7-dev python3-dev

# Download OpenCV
cd ~
wget -O opencv.zip https://github.com/Itseez/opencv/archive/3.3.0.zip
unzip opencv.zip
#Download OpenCV contrib
wget -O opencv_contrib.zip https://github.com/Itseez/opencv_contrib/archive/3.3.0.zip
unzip opencv_contrib.zip

# Download python pip
wget https://bootstrap.pypa.io/get-pip.py
sudo python3 get-pip.py

# Install required python packages
pip install --user numpy "picamera[array]"

# Compile OpenCV
cd ~/opencv-3.3.0/
mkdir build
cd build
cmake -D CMAKE_BUILD_TYPE=RELEASE \
      -D CMAKE_INSTALL_PREFIX=/usr/local \
      -D INSTALL_PYTHON_EXAMPLES=ON \
      -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib-3.3.0/modules \
      -D BUILD_EXAMPLES=ON ..
make
```

2.2 Przechwytywanie i wyświetlanie obrazu z kamery

Po skończonej instalacji i kompilacji, następnym krokiem było przechwycenie obrazu z kamery i wyświetlenie go w oknie. W tym celu użyto następującego kodu:

```
import numpy as np
import cv2

cap = cv2.VideoCapture(0)
```

```
5
6  while(True):
7      ret, frame = cap.read()
8
9      cv2.imshow('frame',frame)
10
11     if cv2.waitKey(1) & 0xFF == ord('q'):
12         break
13
14 # When everything done, release the capture
15 cap.release()
16 cv2.destroyAllWindows()
```

Obraz nie jest w żaden sposób przetwarzany, i kod ten służy tylko do sprawdzenia, że kamera działa.

2.3 Wykrywanie znaczników

Mając obraz z kamery, można było zacząć wykrywać markery. Jako podstawy użyto kody z poprzedniego punktu. Zmodyfikowano go tak, aby wykrywał znaczniki Aruco na przychwyconym obrazie.

Na początku skryptu zadeklarowano dwie zmienne, wymagane przez OpenCV:

```
1 aruco_dict   = aruco.getPredefinedDictionary(aruco.DICT_4X4_50)
2 parameters   = aruco.DetectorParameters_create()
```

Pierwsza zmienna to lista predefiniowanych znaczników, z kolekcji pięćdziesięciu znaczników o wymiarach 4 na 4, druga to domyślne parametry wyszukiwania znaczników.

Następnie ustawiono rozdzielczość, z jaką przechwytywany jest obraz z kamery

```
1 cap = cv2.VideoCapture(0)
2 cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
3 cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
```

W głównej pętli dodano filtr, konwertujący przechwycony obraz na skalę szarości. Konwersja ta jest wymagana przez OpenCV, gdyż znacząco zmniejsza to złożoność obliczeniową tego zadania.

```
1 gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

W uzyskanym w ten sposób obrazie wyszukano wszystkiego, co przypomina znacznik.

```
1     corners, ids, rejected = aruco.detectMarkers(image=gray, dictionary=aruco_dict,
      parameters=parameters)
2
3     if ids is not None:
4         aruco.drawDetectedMarkers(frame, corners)
```

Następnie należy w jakiś sposób wykorzystać znalezione znaczniki. Poniższy fragment kodu oblicza pozycję znacznika, a następnie wypisuje znalezione dane na standardowym strumieniu wyjścia:

```
1     if ids is not None:
2         aruco.drawDetectedMarkers(frame, corners)
3         for i in range(0, len(ids) ):
4                 corner = corners[i][0]
5                 x_sum = corner[0][0] + corner[1][0] + corner[2][0] + corner[3][0]
6                 x = x_sum /4
7                 normalised = (x - 640) / 640
8                 print(ids[i][0], -normalised)
```

Pozycja znacznika jest znormalizowana do zakresu $[-1, 1]$.

W każdej klatce, w której został znaleziony jakiś znacznik, na standardowym wyjściu wypisywana jest informacja w formacie `"[id_znacznika] [pozycja_x]"`

Skończony, działający kod wygląda następująco:

```
import numpy as np
import cv2
import cv2.aruco as aruco
import sys, time, math

aruco_dict  = aruco.getPredefinedDictionary(aruco.DICT_4X4_50)
parameters  = aruco.DetectorParameters_create()

cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)

while True:

    ret, frame = cap.read()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    corners, ids, rejected = aruco.detectMarkers(image=gray, dictionary=aruco_dict,
    parameters=parameters)

    if ids is not None:
        aruco.drawDetectedMarkers(frame, corners)
        for i in range(0, len(ids) ):
            corner = corners[i][0]
                x_sum = corner[0][0] + corner[1][0] + corner[2][0] + corner[3][0]
                x = x_sum / 4
                normalised = (x - 640) / 640
                print(ids[i][0], -normalised)

    cv2.imshow('frame', frame)

    # Nacisniecie klawisza q zatrzymuje program
    key = cv2.waitKey(1) & 0xFF
    if key == ord('q'):
        cap.release()
        cv2.destroyAllWindows()
        break
```

3 Napotkane problemy

3.1 Dokładność wykrywania znaczników

Z powodu braku plików z kalibracją kamery, program często ”gubi” znaczniki, i nie jest w stanie wykrywać ich ze stuprocentową skutecznością.

3.1.1 Możliwe rozwiązanie

Używając biblioteki OpenCV możliwe jest napisanie programu, który wygeneruje niezbędne informacje kalibracyjne. Następnie można wczytać te informacje do programu i przekazać je jako parametry funkcji `aruco.DetectMarkers`.

3.2 Dane nie są wysyłane przez port szeregowy

Ze względu na brak czasu, nie udało się zaimplementować wysyłania wynikowych informacji przez port szeregowy.

3.2.1 Możliwe rozwiązania

Możliwe są dwa rozwiązania. Pierwsze z nich nie wymaga modyfikacji kodu, a jedynie przekierowanie standardowego wyjścia programu na port szeregowy za pomocą Unixowego operatora przekierowania "`>`". Drugie rozwiązanie to zamiana funkcji "`print`"" w 28 linijce kodu na funkcję, która wyśle dane na port szeregowy.