ApelPro Járgány

Farkas Attila

Tartalomjegyzék

Kitűzött célok

Az autó az IBControll által szervezett Raspberry-s versenyre készült. A projekt fő célja tehát, hogy az autót az említett eszköz vezérelje oly módon, hogy mindig a legoptimálisabb irányba irányítsa az autót. A felmerülő akadályok észlelése, ezekre kikerüléssel, lassítással, megállással, szükség szerint tolatással reagálni.

További célok:

- Robot Operating System megismerése, használata
- Képfeldolgozás alkalmazás a Raspberry beépített kamerája által készített képen valós időben az útvonal intelligens megválasztásához
- Az autó állandó mozgásban tartása, ha megállása nem indokolt: a képfeldolgozás idejére ne álljon meg útvonalat tervezni.
- Az autó minél kisebb rongálásával minél esztétikusabb jármű építése, átalakítása.

Az autóról

Az autót Raspberry Pi2 vezéri, amelyen Robot Operating System segítségével a vezérlés egyes elemei külön egységekben foglalnak helyet:

1. Az autóban található motorokat egy Arduino vezérli, illetve a szenzorok értékeit és az akkumulátor feszültségét is ez olvassa le. Az eszköz ROSSeriallal illeszkedik a főmodulhoz.
2. Az Arduinot vezérlő egység a Raspberryn fut (ArduinoController).
3. A képfeldolgozást végző egység szintén a Raspberryn fut (Capture).

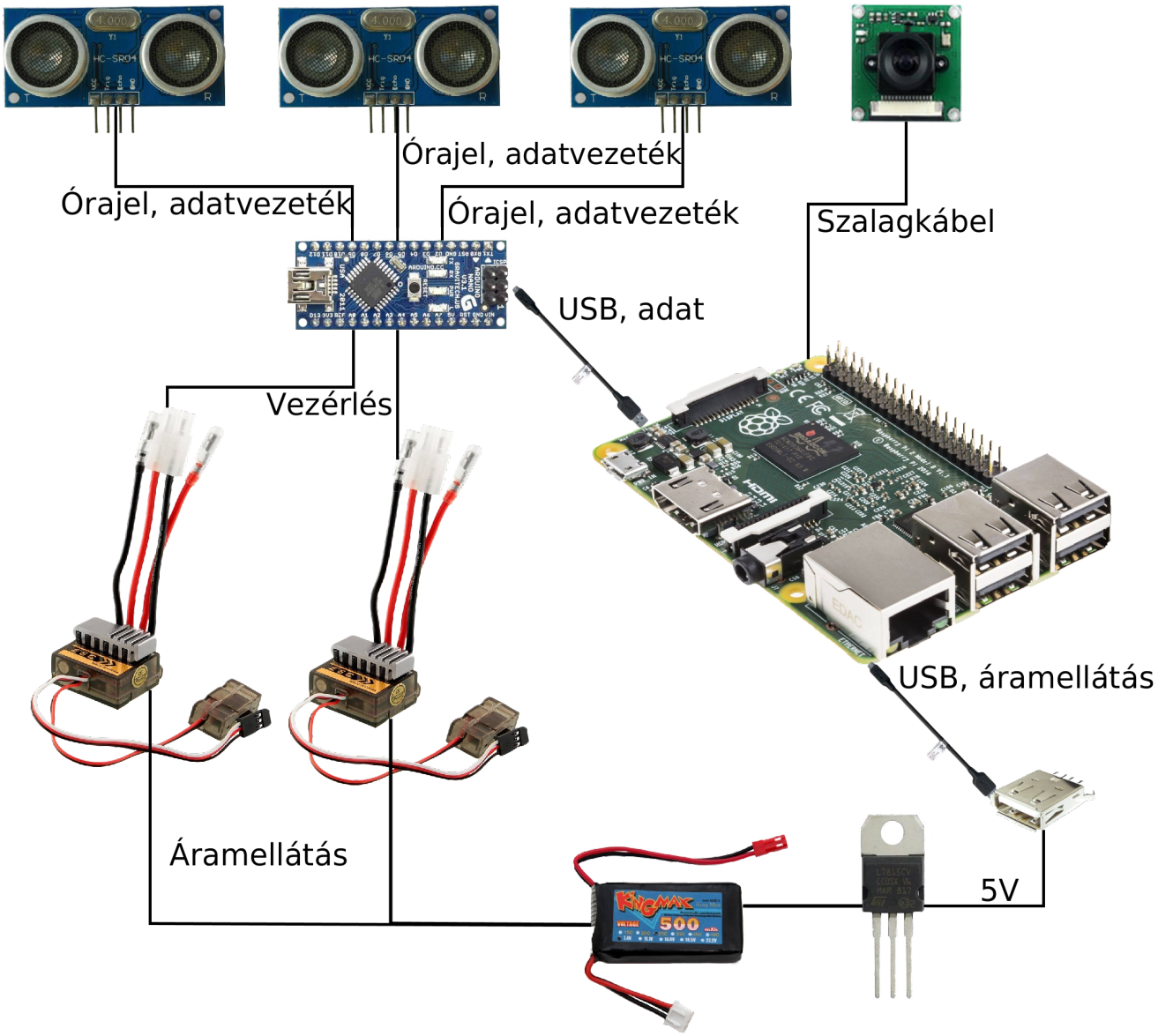
Az autón három távolság (ultrahangos elven működő) érzékelő található, egy a baloldalon, egy középen és egy jobb oldalon. Az autó ablaka el lett távolítva, hogy a kamera kiláthasson az autó belsejéből. Így a következő szembetűnő elem a kamera, mellyel képfeldolgozás után (részben) az útvonal intelligens megválasztása válik lehetővé. E négy szenzor segítségével próbálkozik kikerülni a felbukkanó akadályokat, illetve amennyiben nem képes erre megáll.

Az autó egy Lipo két cellás (7.4V, 1650mAh) akkumulátor segítségével működik, mely egy … (5V, 3A) feszültségstabilizátoron keresztül látja el a Raspberryt.

Az autóban található kettő motorvezérlő (ESC) melyek a hagyományos kefés motorok sebesség változtatását, illetve a hagyományos kefés motorral működő, kanyarodásért felelős motor vezérlését segíti. A motorok közvetlen az akkumulátorra csatlakoznak. A motor védelmében a PWM maximalizálása szükséges, hogy az 5V-os motor ne kapjon nagyobb feszültséget.

Az akkumulátor lemerülésének jelzésére egy Buzzer található az autó eredeti akkumulátorának helyén, az autó aljában.

Az autó eszközei és kapcsolatuk



Robot Operating System

A Robot Operating System (ROS) egy robotok programozására és felügyeletére kifejlesztett könyvtárcsomag Linux-ra. Több különböző szoftvercsalád létezik, a Raspberryn a ROS Indigo fut, mivel egyes fórumok alapján ez támogatja legjobban a képfeldolgozáshoz szükséges egységeket.

A ROS-ban több modult bírunk létrehozni, melyek csatornákon keresztül kommunikálnak/kommunikálhatnak egymással. Egy csatornán kétféle, úgynevezett Node lehet: aki publikál rajta és aki fel van rá iratkozva (publisher és subscriber).

Felépítését tekintve áll egy főmodulból (ROSCore) és az almodulokból, melyek először a főmodulhoz fordulnak, megosztják csatornáik nevét melyen publikálnak és lekérik azon modulok nevét, melyekre azok publikálnak, amelyekre fel van iratkozva. Ezek után az egyes modulok pont-pont kapcsolattal vannak összekötve, vagyis a továbbiakban a kommunikáció nem a főmodulon keresztül zajlik. Az egyes modulok párhuzamosan, „egyszerre” futnak akár több hárdveren is.

Megjegyzés:

Utóbbi a projektben is fellelhető: a Raspberryn fut a főmodul, míg az Arduino ROSSeial segítségével kapcsolódik hozzá USB kábelen, és fogadja, illetve küldi az adatokat.

ROSSerial

A hardverelemek összeköttetésére alkalmas modul ROS-ban. Egy modul, melyet elindítva és felparaméterezve hozzáadhatjuk USB-vel, vagy ethernet kábellel csatlakoztatott hárdvereinket, melyeken ROS program fut. A csatlakoztatott eszközön nem futhat főmodul (ROSCore), mivel egy rendszerben egy ilyen lehet. Ezzel a modullal generálhatjuk le az Arduino használatához szükséges könyvtárat, melyet a fejlesztőbe beemelve példaprogramokat is kapunk.

ROS programozás

#!/usr/bin/env python
Futási környezet beállítása

import rospy
from std_msgs.msg import String
from std_msgs.msg import UInt16
Függvénykönyvtárak behívása

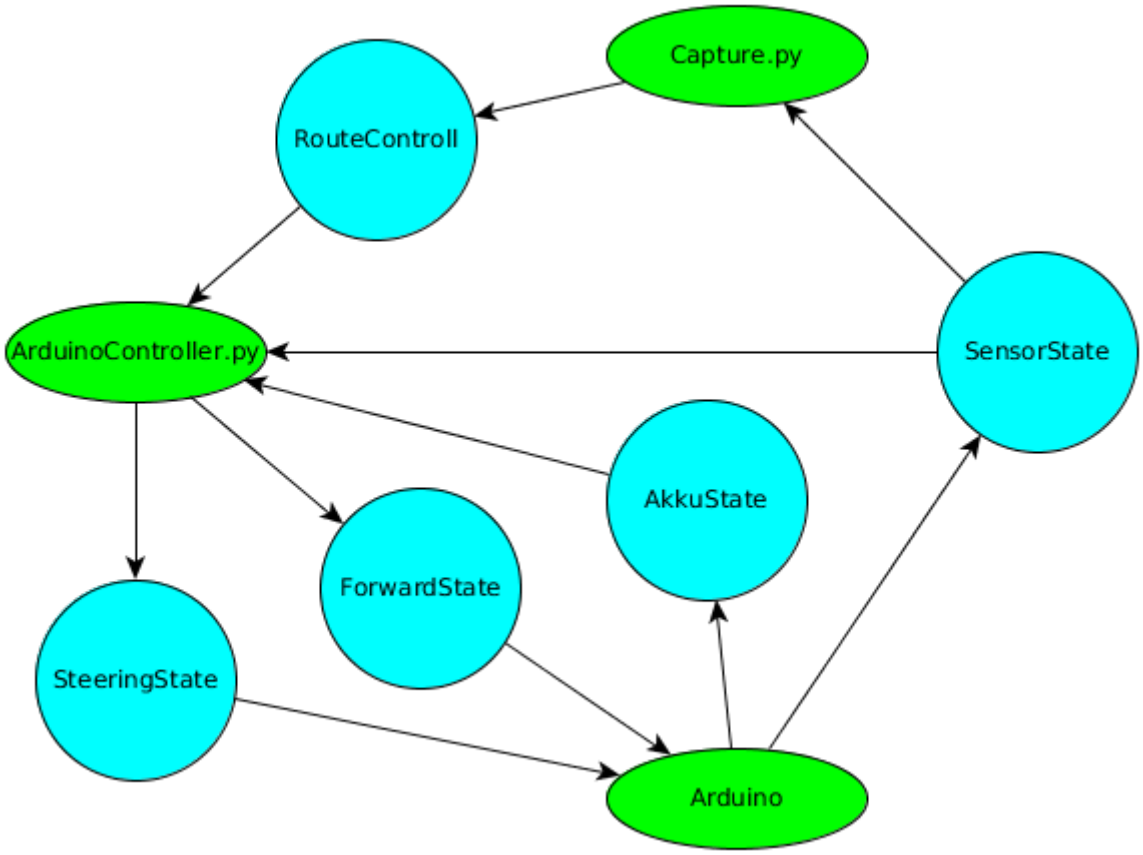
publish=rospy.Publisher("teszt",UInt16,queue_size=10)
Publikáló létrehozása

rospy.init_node("teszt_ROS",anonymous=True)
ROS inicializálása
publish.publish()
Publikálás a publish által meghatározott csatornán

```
def Function(data):
    rospy.loginfo(data.data)
```
Kapott adat kiírása képernyőre időegységekkel együtt

```
def Subscriber():
    rospy.Subscriber("teszt",UInt16,Function)
```
Feliratkozó definiálása

ROS modulok a projektben

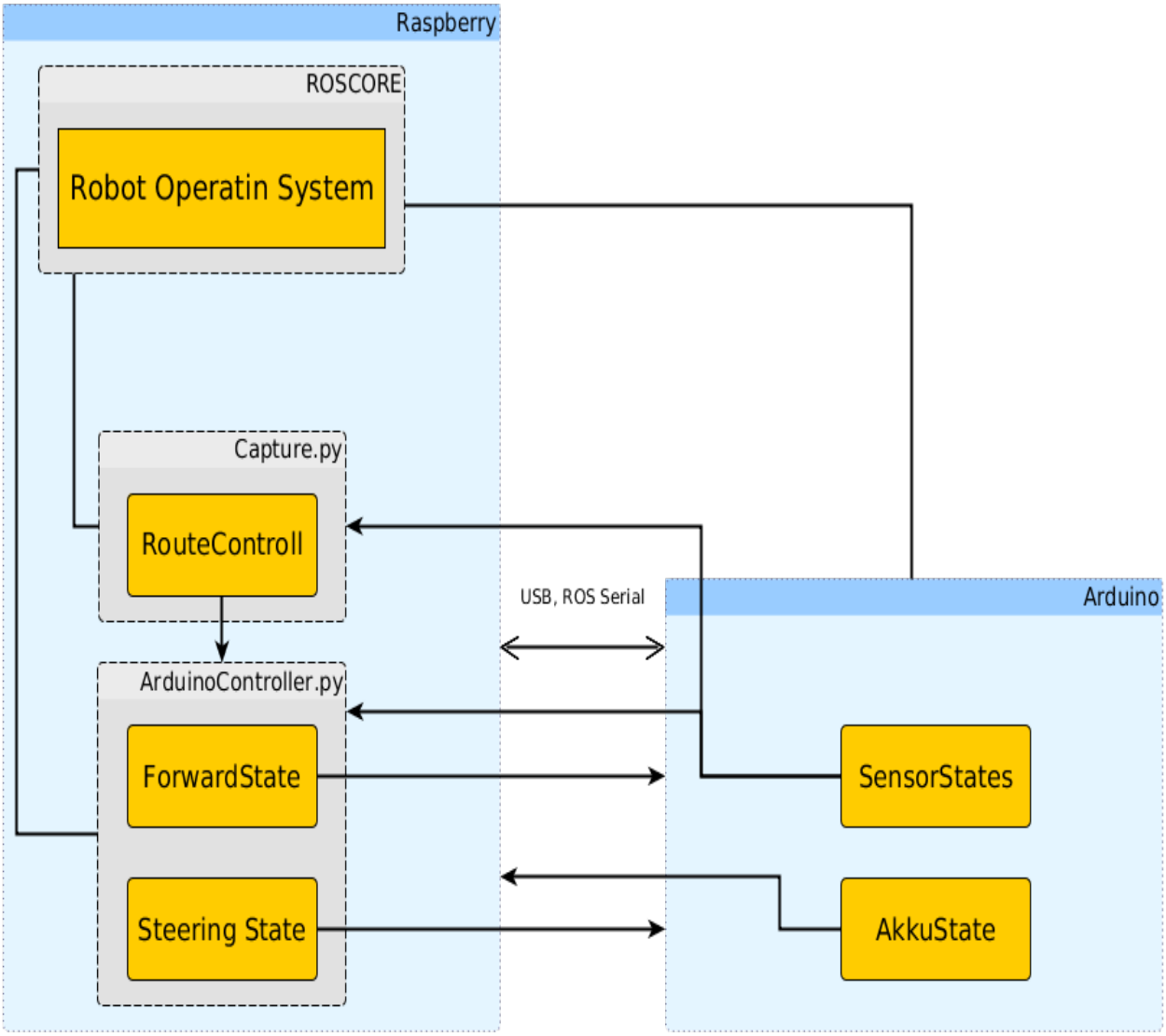


A nyilak az adatáramlási irányba mutatnak

Leírás:

- Az Capture.py publikál a 'RouteControll' csatornán, fel van iratkozva a 'SensorStates'-re.
- Az ArduinoController.py publikál a 'ForwardState' és a 'SteeringState' csatornákon, fel van iratkozva a 'SensorStates', az 'AkkuState' és a 'RouteControll' csatornákra.
- Az arduino publikál a 'SensorStates' és a 'AkkuState' csatornákon, fel van íratkozva a 'ForwardState' és a 'SteeringState' csatornákra.

ROS modulok másképp



A képek az adatáramlás irányába mutatnak

A kép az egyes modulok hardveres elhelyezkedését szemlélteti. A kék bekeretezett részek a hardvereket jelölik, a szürke bekeretezett részek egy-egy modult jelölnek, a narancssárga téglalapok a csatornát, amelyen publikálnak.

Capture.py

Ebben a modulban készül a kép és egyből fel is dolgozza azt. Raspberryn sajnos nincs lehetőség, illetve hosszú úton juthatunk el addig, hogy videó felvételt készíthessünk (OpenCV-ben a VideoCapture USB-s kamerával dolgozik alapértelmezetten: raspberryhez kapcsolt USB-s kamerával működik, a belső kamerához telepíteni kell kiegészítőket); a képek egyszeri felvétele pedig a kamera bemelegedésére várakozás miatt mozgás szempontjából sok időbe telik. Ezért sorozatfelvétel készül, melynek során a kamera nem kapcsol ki a kép elkészítése után, a képkockákat pedig fel lehet dolgozni. Átalakítások nélkül a kb. 10-15 képkocka/másodperces sebességet produkál, feldolgozással 7-8 képkocka/másodpercre csökkent.

Feldolgozás módja:

A készített képből először szürkeárnyalatos kép készül, majd az így kapott kép maszkolással fekete-fehérré alakul, melyben lehet kontúrvonalakat keresni. A megkapott kontúrvonalakból a leghosszabbat megkeresve, adott a legnagyobb fehér folt a képen.
A képből kiszűrve az autó által kitakart részt, s az így kapott képet négy részre osztva, a megkeresett kontúrvonalon végighaladva megállapítható a négy kritikus pont: a baloldali képeknek a legjobb oldalibb pontja, és a jobb oldaliaknak a legbaloldalibb pontja.

A kapott koordináták és a szenzorok értékei alapján megállapíthatja, hogy haladhat-e tovább, illetve merre van lehetősége tovább haladni.
A lehetséges irányok közül az dönt, hogy merre van messzebb az adott akadály, vagyis a távolságérzékelők alapján dönt.

A folyamat végén kiválasztott irányt küldi el a 'RouteControll' csatornán, de csak akkor, ha az előző állapothoz képest változás történt. Ellenkező esetben nem küld semmit.

ArduinoController.py

Figyeli, hogy az autó nem került-e közel bármi akadályhoz a középső távolságérzékelő alapján. Ha ez megtörtént, akkor megállítja a motort és ha egyébként nem állt, akkor elküldi azt az Arduinonak.

Ezután a 'RouteControll' csatornán kapott irányok és a szenzorok értékei alapján cselekszik. Lekérdezi mely irányba kell mennie a kép alapján, majd az adott irányba néző szenzor értéke szerint megállapítja, hogy milyen sebességgel tudja ezt végrehajtani. Egy bizonyos távolságban lelassul, ha túl közel ért meg áll. Egyéb esetben konstans sebességgel halad.

A kapott irányt a 'SteeringState' csatornán küldi el ha különbözik az előzőektől, míg a sebességet a 'ForwardState'-en teszi közzé, ugyancsak ha változott az előzőhöz képest.

A kettőnek egymás után azt (kellene) eredményeznie, hogy amennyiben van valami előtte, de a képfeldolgozás alapján mehet, úgy lassú tempóban kerüli ki a szóban forgó akadályt.

Mindeközben fel van iratkozva az 'AkkuState'-re, melyen AkkuFail üzenetek folyamatos megjelenése esetén egy továbbfejlesztési lehetőségként leállítja a Raspberryt az akkumulátor és saját maga védelmében.

Arduino

Az arduinora a már említett ROS könyvtárral írhatunk programot. A helyes működéshez egy-két parancs kötelező, de ezentúl úgy alkalmazhatjuk rendszerünkben, mint bármely más esetben.

Először is megpróbál egy sebességet ('ForwardState') és egy irányt ('SteeringState') lekérdezni, és ha kapott információt, hogy hogyan/merre kell mennie, akkor végrehajtja azt. A sebességet százalékban kapja, míg a kanyarodásnál egyetlen dolgot kell eldönteni, hogy balra, vagy jobbra megy, nem állítható adott szögre.

Következő lépésben szoftveres megszakítás elvén lekérdezi a szenzorok értékeit. A középsőt egyből, míg a két szélsőt egymás után 10ms-os, a középsőhöz képest 100ms-os késleltetéssel kérdezi le. Így az érzékelők nem zavarják meg egymást. A szenzorokat egységesen 50ms-ként vizsgálja. A lekérdezés után egyből el is küldi az értékeket a 'SensorStates' csatornán, a középsőt nyers adatként, a jobb oldalit 20000-el, a bal oldalit 40000-el eltolva.

Majd végül az akkumulátor állapotát kéri le. Két cella rákötése esetén mind a kettőt vizsgálja, ha az egyik 2.9V vagy azalatt van, akkor elkezd sípolni, és küld egy üzenetet az 'AkkuState' csatornán, hogy merül az akkumulátor: „AkkuFail”.

Megvalósított célok, továbbfejlesztési lehetőségek

A célokat sikerült megközelíteni, és nagyobb részük sikeresen teljesítve lett.

A képfeldolgozás csak részben teljesített, kevés intelligenciát ad a járműnek. A kép alaposabb feldolgozása orvosolhatná ezt a problémát.

A hátramenet, s így a fékező hatás is elveszett az egyik motorvezérlő hibája kapcsán. Így a szenzorokat kellően magas értéken figyelembe kell venni ahhoz, hogy lendületből is meg tudjon állni. Alacsony sebességen hamar meg áll, ekkor még el bírna kanyarodni.

Továbbfejlesztési lehetőségként megemlíteném a Buzzer kihasználásának lehetőségét, illetve az Arduino által küldött 'AkkuFail' üzenet figyelését, amely az akkumulátor lemerülését jelzi. Az üzenet beérkezésekor a Raspberryt le kell állítani az adatvesztés, és az akkumulátor meghibásodásának elkerülése érdekében.

Automatikus indítás, hogy a rendszerrel együtt a ROS is elinduljon, így az autó is.

Gomb felszerelése, mellyel az autó működése aktiválható/deaktiválható.

Mindkét akkumulátor cella mérése a pontosabb és biztonságosabb mérés érdekében.

Az autó jelenleg „random felderítő„-ként működik. Pontosabb érzékelőkkel és egy új motorvezérlővel hátul akár irányított felderítővé is válhat, amelyhez azonban egy a kerékforgását visszamérő tárcsa, illetve a hozzá szükséges optikai kapu is szükséges lehet, melyről a tervezés során skicc szerű terv készült.

Források

Robot Operating System:	http://wiki.ros.org/
OpenCV:	http://www.pyimagesearch.com
	http://raspberrypi.stackexchange.com

A projekthez szükséges fájlokat a github-ról töltöttem le: https://github.com/
Teljes linkek:
http://wiki.ros.org/ROSberryPi/Installing%20ROS%20Indigo%20on%20Raspberry%20Pi
http://www.pyimagesearch.com/2014/10/20/finding-shapes-images-using-python-opencv/
http://raspberrypi.stackexchange.com/questions/24262/getting-image-data-from-raspberry-pi-camera-module-in-opencv-with-python