PROJECTION BASED PICKING SYSTEM

Introduction

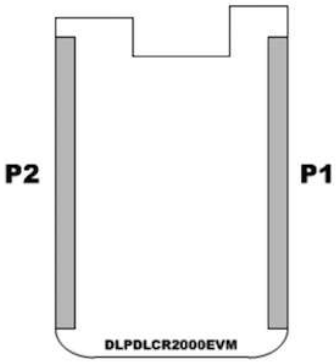
Projection based picking system is order Picking System which is one of the most important tasks in modern warehouses. Projection based picking system can be employed in warehouses where the sizes of containers or drawers is less than than the length of the PTL. This new system works with a projector mounted at position appropriate to cover the entire rack of the warehouse. Cameras are installed at end of the rack so that it captures the special Tag attached at the side face of the drawers when it is opened and accordingly sends the information to the controlling system.

Excel sheets are uploaded and downloaded in a folder. The sheet contains the item name and the quantity to be picked. The projector projects a yellow blinking light on the drawer which contains the item. Individual navigates to the drawer, opens it and picks the mentioned quantity. A green light on drawer marks a successful pick. A wrong open of drawer makes the projector project a red light on the entire rack. The lists completed moves to a completed job folder and the process repeats.

The current system comprises of DLP EVM 2000 projector connected to a Raspberry Pi (RPi1). Another Raspberry Pi (RPi2) is connected with a USB Camera which identifies the AprilTag ID. RPi2 sends the data to RPi1 which performs the comparison actions and projects the light on the drawer.

Circuit

P2	
1: GND	2: GND
3: VINTF	4: N/A
5: N/A	6: N/A
7: N/A	8: N/A
9: N/A	10: N/A
11: N/A	12: N/A
13: N/A	14: N/A
15: PROJ_ON_EXT	16: N/A
17: N/A	18: N/A
19: EXT_SCL	20: EXT_SDA
21: N/A	22: N/A
23: N/A	24: N/A
25: N/A	26: N/A
27: N/A	28: N/A
29: N/A	30: N/A
31: N/A	32: N/A
33: N/A	34: N/A
35: N/A	36: N/A
37: N/A	38: N/A
39: N/A	40: N/A
41: N/A	42: N/A
43: HOST_PRESENTZ	44: GND
45: GND	46: GND



Legend:
Not Used
Parallel I/F Video Data
Parallel I/F Video Control
I2C Bus
System Supply Lines
System Control Lines

P1	
1: N/A	2: N/A
3: N/A	4: N/A
5: N/A	6: N/A
7: N/A	8: N/A
9: N/A	10: N/A
11: Data18	12: Data19
13: Data22	14: Data21
15: Data16	16: Data17
17: Data20	18: GPIO5
19: Data23	20: N/A
21: N/A	22: N/A
23: N/A	24: N/A
25: N/A	26: GPIO_INIT_DONE
27: VSYNC	28: PCLK
29: HSYNC	30: DATAEN
31: Data14	32: Data15
33: Data13	34: Data11
35: Data12	36: Data10
37: Data8	38: Data9
39: Data6	40: Data7
41: Data4	42: Data5
43: Data2	44: Data3
45: Data0	46: Data1

Connections on DLP EVM 2000

1		P2_6: 5V	
	P1_27: VSYNC		
	P1_29: HSYNC		
	P1_16: Data17	P1_38: Data9	
		P1_36: Data10	
	P1_45: Data0	P1_46: Data1	
		P2_20: EXT_SDA	
		P2_19: EXT_SCL	
	P1_42: Data5		
	P1_32: Data15		
	P1_39: Data6	P1_31: Data14	
		P1_33: Data13	
	P1_28: PCLK	P1_30: DATAEN	
	P1_34: Data11		
	P1_35: Data12	P1_40: Data7	
	P1_37: Data8		
	P1_43: Data2	P1_15: Data16	
		P1_44: Data3	
	P2_46: GND	P1_41: Data4	40

Connections on RPi1

The below configuration needs to be appended at the end of the config.txt file of RPi1:

```
# Add support for software i2c on gpio pins
dtoverlay=i2c-gpio,i2c_gpio_sda=23,i2c_gpio_scl=24,i2c_gpio_delay_us=2

# DPI Video Setup
dtoverlay=dpi18
overscan_left=0
overscan_right=0
overscan_top=0
overscan_bottom=0
```

```
framebuffer_width=854
framebuffer_height=480
enable_dpi_lcd=1
display_default_lcd=1
dpi_group=2
dpi_mode=87

dpi_output_format=458773
hdmi_timings=854 0 14 4 12 480 0 2 3 9 0 0 0 60 0 32000000 3
```

To display the Pi's Desktop the following commands need to be passed and should be appended in /etc/rc.local before the exit 0 command.

```
sudo i2cset -y 3 0x1b 0x0c 0x00 0x00 0x00 0x13 i
sudo i2cset -y 3 0x1b 0x0b 0x00 0x00 0x00 0x00 i
```

AprilTags

AprilTags are new visual fiducial system that uses a 2D bar code style ``tag'', allowing full 6 DOF localization of features from a single image.

The algorithm has nine steps:

1. Convert the image to floating point grayscale (pixel values between 0.0 and 1.0) and apply a Gaussian blur.
2. Calculate the local gradient (magnitude and direction) at each pixel.
3. Generate a list of edges, grouping connected pixels with similar directions together. An edge is present if the magnitude of the gradient for both pixels is significantly above zero.
4. Create clusters from the edges.
5. Loop over the clusters, fitting lines called Segments.
6. For each Segment, find segments that begin where this segment ends.
7. Search all connected segments to find loops of length 4, called Quads. Each quad represents the black border around a tag candidate.
8. Decode the quads by looking at the pixels inside the border to see if they represent a valid tag code, and generate a list of TagDetections
9. Search for overlapping TagDetections and take the best ones (lowest Hamming distance or largest perimeter); discard the rest.

Python based package for AprilTag Detection was used for the identification of the AprilTag ID. The link to package is https://pypi.org/project/apriltag/ or it can be installed by running "**pip install apriltag**". Ensure that **CMAKE** is installed first. Also OpenCV is required for image and video handling.

Below is the code which detects the apriltag, identifies the ID and then writes the ID in a .txt file. The following code is run on RPi2 with USB Camera connected to it.

```
#!/usr/bin/env python

'''Demonstrate Python wrapper of C apriltag library by running on camera frames.'''
from __future__ import division
from __future__ import print_function

from argparse import ArgumentParser
import cv2
import apriltag

# for some reason pylint complains about members being undefined :(
# pylint: disable=E1101

global x

def main():

    '''Main function.'''

    parser = ArgumentParser(
        description='test apriltag Python bindings')

    parser.add_argument('device_or_movie', metavar='INPUT', nargs='?', default=0,
                        help='Movie to load or integer ID of camera device')

    apriltag.add_arguments(parser)

    options = parser.parse_args()

    try:
        cap = cv2.VideoCapture(int(options.device_or_movie))
    except ValueError:
        cap = cv2.VideoCapture(options.device_or_movie)

    window = 'Camera'
    cv2.namedWindow(window)

    # set up a reasonable search path for the apriltag DLL inside the
    # github repo this file lives in;
    #
    # for "real" deployments, either install the DLL in the appropriate
    # system-wide library directory, or specify your own search paths
    # as needed.

    detector = apriltag.Detector(options,
                                 searchpath=apriltag._get_demo_searchpath())

    f = open("test.txt","w+")
    f.write('0')
```

```
    f.close()

    while True:

        success, frame = cap.read()
        if not success:
            break

        gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
        detections, dimg = detector.detect(gray, return_image=True)

        num_detections = len(detections)
#        print('Detected {} tags.\n'.format(num_detections))
        if num_detections > 0:
      for detection in enumerate(detections):
         print(detection[1][1])
         x = detection[1][1]
      f = open("test.txt","w+")
      f.write(str(x))
      f.close()


#        for i, detection in enumerate(detections):
#            print('Detection {} of {}:'.format(i+1, num_detections))
#            print()
#            print(detection.tostring(indent=2))
#            print()

        overlay = frame // 2 + dimg[:, :, None] // 2

        cv2.imshow(window, overlay)
        k = cv2.waitKey(1)

        if k == 27:
            break

if __name__ == '__main__':
    main()
```

GUI for Projection

The GUI for projection is run on RPi1 which is connected to the projector. The GUI is implemented using the PyQt library of python. The design file of GUI is flexible such that it can be changed based on the number of drawers or the rack size.

The design file for the GUI is

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'pyqt.ui'
#
# Created by: PyQt4 UI code generator 4.12.1
#
# WARNING! All changes made in this file will be lost!

from PyQt4 import QtCore, QtGui
```

```
try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    def _fromUtf8(s):
        return s

try:
    _encoding = QtGui.QApplication.UnicodeUTF8
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig, _encoding)
except AttributeError:
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig)

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName(_fromUtf8("MainWindow"))
        MainWindow.resize(553, 500)
        self.centralwidget = QtGui.QWidget(MainWindow)
        self.centralwidget.setObjectName(_fromUtf8("centralwidget"))


        self.Boxes = []
        for i in range(8):
            self.Boxes.append(QtGui.QLabel(self.centralwidget))


        for i in range(4):
            self.Boxes[i].setGeometry(QtCore.QRect(60+i*120, 80 , 100, 100))
            self.Boxes[i].setAlignment(QtCore.Qt.AlignCenter)
            self.Boxes[i].setObjectName(_fromUtf8("Box" + str(i+1)))
            self.Boxes[i].setStyleSheet("QLabel{background-color : white;}")

        for i in range(4):
            self.Boxes[i+4].setGeometry(QtCore.QRect(60+i*120, 220 , 100, 100))
            self.Boxes[i+4].setAlignment(QtCore.Qt.AlignCenter)
            self.Boxes[i+4].setObjectName(_fromUtf8("Box" + str(i+5)))
            self.Boxes[i+4].setStyleSheet("QLabel{background-color : white;}")

        self.pushButton = QtGui.QPushButton(self.centralwidget)
        self.pushButton.setGeometry(QtCore.QRect(110, 420, 97, 27))
        self.pushButton.setObjectName(_fromUtf8("pushButton"))


        self.quantity = QtGui.QLabel(self.centralwidget)
        self.quantity.setGeometry(QtCore.QRect(70, 350, 201, 41))
        self.quantity.setStyleSheet(_fromUtf8(""))
        self.quantity.setAlignment(QtCore.Qt.AlignCenter)
        self.quantity.setObjectName(_fromUtf8("quantity"))
        self.label = QtGui.QLabel(self.centralwidget)
        self.label.setGeometry(QtCore.QRect(16, 30, 311, 41))
        self.label.setAlignment(QtCore.Qt.AlignCenter)
        self.label.setObjectName(_fromUtf8("label"))

        MainWindow.setCentralWidget(self.centralwidget)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow", None))
        self.pushButton.setText(_translate("MainWindow", "next", None))
        for i in range(8):
```

```
            self.Boxes[i].setText(_translate("MainWindow", "Box"+str(i+1), None))
        self.quantity.setText(_translate("MainWindow", "Quantity to be Picked", None))
        self.label.setText(_translate("MainWindow", "Status", None))
```

Functions for the backend working of the Projection system:

```
class Functions:

    def flashLbl(self,i):
        if self.lblWhite == True:
            self.Boxes[i].setStyleSheet("QLabel{background-color : yellow;}")
            self.lblWhite = False
        else:
            self.Boxes[i].setStyleSheet("QLabel{background-color : white;}")
            self.lblWhite = True

    def go(self, x):
        self.Boxes[x-1].setStyleSheet("QLabel{background-color : green;}")


    def wrong(self):
        for i in range(8):
            self.Boxes[i].setStyleSheet("QLabel{background-color : red;}")



    def reset(self):
        for i in range(8):
            self.Boxes[i].setStyleSheet("QLabel{background-color : white;}")
```

The main file for system is:

The following packages are required :

1. Time – for delay and sleep
2. Xlrd – for using excel files
3. Os – for navigating in directories
4. Shutil – moving files in folders

```
from PyQt4 import QtGui, QtCore
import sys
import design_new
import time
import xlrd
import functions
import os
import shutil
```

```
sys.setrecursionlimit(1500)

class ExampleApp(QtGui.QMainWindow, design_new.Ui_MainWindow, functions.Functions):
   def __init__(self):
      self.flag = 1
      self.folder_path = "/home/pratik/Documents/PyQt/PickProjection/in/"

      super(self.__class__, self).__init__()
      self.setupUi(self)
      self.pushButton.clicked.connect(self.next)

   def check(self,x,y):

      if y==x:
         self.label.setText("Good Job")
         self.quantity.setText("Pick")
         self.go(x)
         QtCore.QCoreApplication.processEvents()
         time.sleep(2)
         self.reset()
         QtCore.QCoreApplication.processEvents()
         self.i += 1

         if self.i < self.len_sheet:
            self.next()
         else:
            self.label.setText("All items of the sheet completed")
            self.flag = 1
            shutil.move(self.folder_path + self.list[0],
"/home/pratik/Documents/PyQt/PickProjection/out/" + self.list[0])
            self.next()


      else:
         self.label.setText("Pick from the Yellow Blinking Box")
         self.wrong()
         QtCore.QCoreApplication.processEvents()
         time.sleep(2)
         self.reset()
         QtCore.QCoreApplication.processEvents()
         prev_y = y

         while y == prev_y:

            self.Boxes[x-1].setAutoFillBackground(True)

            self.flashLbl(x-1)
            QtCore.QCoreApplication.processEvents()

            f = open("test.txt", "r")
            data = f.read()
            y = int(data)
            f.close()

            QtCore.QCoreApplication.processEvents()
            time.sleep(0.25)

         self.check(x,y)

   def hibernate(self):
      time.sleep(5)
      self.flag = 1
      self.next()
```

```
   def next(self):

      if self.flag == 1:
         self.list = (os.listdir(self.folder_path))
         if len(self.list) == 0:
            self.hibernate()

         else:
            self.wb = xlrd.open_workbook(self.folder_path + self.list[0])
            self.sheet = self.wb.sheet_by_index(0)
            self.i = 0
            self.len_sheet = self.sheet.nrows

      self.flag = 0

      f = open("test.txt", "w")
      f.write('0')
      f.close()

      x = (int(self.sheet.cell_value(self.i, 0)))
      pick = (int(self.sheet.cell_value(self.i, 1)))
      self.label.setText("Pick from the Blinking Box")
      self.quantity.setText("Pick {} Units".format(pick))

      f = open("test.txt", "r")
      data = f.read()
      y = int(data)
      f.close()

      self.lblWhite=True

      while y == 0:
         self.Boxes[x-1].setAutoFillBackground(True)

         self.flashLbl(x-1)
         QtCore.QCoreApplication.processEvents()

         f = open("test.txt", "r")
         data = f.read()
         y = int(data)
         f.close()

         QtCore.QCoreApplication.processEvents()
         time.sleep(0.25)

      self.check(x,y)

      return



def main():
   app = QtGui.QApplication(sys.argv)
   form = ExampleApp()
   form.show()
   app.exec_()

if __name__ =='__main__':
   main()
```

References:

1. https://pypi.org/project/apriltag/
2. http://frederickvandenbosch.be/?p=2948
3. http://e2e.ti.com/support/dlp/f/94/p/651931/2405155#2405155?jktype=e2e
4. https://pinout.xyz/pinout/dpi
5. https://www.element14.com/community/roadTestReviews/2662/l/dlp-pico-display-projector-evm-beaglebone-black-review
6. http://www.ti.com/lit/ug/dlpu049c/dlpu049c.pdf