Department of Electrical Engineering and Computer Engineering
Howard University
Washington, DC 20059

HOWARD UNIVERSITY

Terminator

By:
Charles Robinson (EE)| Owens Vil(EE)|
Marcus Ragland(EE)| Chibuike Agba(ME)|
Tekevwe Akoroda(ME)

Dr. Kim
4/25/2019

Table of Contents

Introduction and Background

The purpose of this project to build an intelligent robot that implemented in tic tac toe games. This robot will understand the rules and ways to play the game and make every decision based on the intelligence of itself. To test the intelligence of this robot, the opponent must be a human. This project is primarily focused on the feasibility of a AI-based game playing system, and if one can be created cheaply.

In designing the Terminator, each EE group member brought forward an original drawing of how they think the system should be designed. Of these designs, the team decided to narrow down selection to just two designs for which the Terminator may be based upon. From these top two designs, the group weighed their options using a design matrix in order to tell, by scores, which design would be followed. Every drawing and idea had its own pros and cons, so, instead of just choosing one single design, the team found ways to incorporate ideas and concepts from each other's designs to make one final design.

Cost

The parts required for the Arms assembly average at around $40 each. The Robotic Arm Edge, which is the base structure costs $40, while the Raspberry Pi is $49.99. The webcam for OpenCV costed $29.99. Total cost is vastly cheaper than expected.

Problem Statement

To design, build and integrate the elements of an autonomous robotic system that plays Tic -Tac -Toe on a 3 by 3 grid.

Long Term Goal

To develop an autonomous tic-tac-toe playing robot that competes against a human opponent.

2018-2019 Year Goal

To program the Raspberry Pi to give the capability to recognize 3x3 tic tac toe grid.

We intend to produce an AI robot using mainly existing components such as the Robotic Arm Edge, the Raspberry Pi, the Arduino, and a circuit-board camera compatible with the Raspberry Pi. All the components interface with each other using OpenCV and Python.

Design Requirements

There are essential, proposed vision related, energy/power/environmental, size & weight, and gameplay requirements.

Essential Requirements:

The robot must have a perception of the environment, the capability to evaluate moves based on the state of the board, know the rules of tic tac toe, the ability to grip tic tac toe pieces(1-4 cm in width), and reach every location in playing space.

Proposed Vision Related Requirements:

The camera stand shall be 18 cm above the game board, while not weighing more than 1Kg and being manufactured at Howard University.

Energy, Power, & Environmental Requirements:

The robot must be powered by 9V batteries.

Size & Weight Requirements:

The robot arm should weigh less than 1Kg, while the camera stand and board should weigh no more than 2Kg.

Gameplay (Control) Requirements:

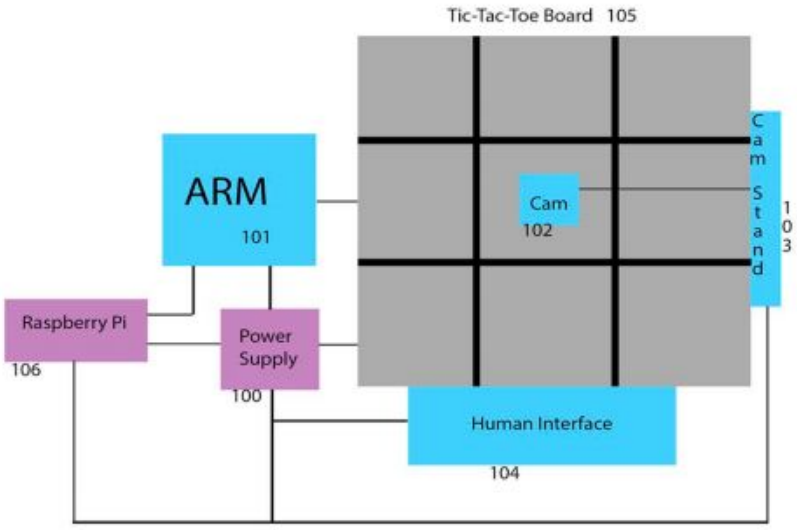
Execution of play should take less than 20 seconds, while stabling motion with minimal or no vibrations/oscillations.

Standards & Regulations:

The robotic arm must comply with OSHA 29 CFR 1910.333 (selection and use of work practices) and OSHA OSHA 29 CFR 1910.147 (the control of hazardous energy)

Solution Design:

Top Solution Design



Solution Design Description

The team agreed on the top design where, the power supply **100** provides power to the initial system in order to power the robot arm **101**, camera **102**, driver-vehicle interface **104**, and raspberry pi **106**. The raspberry pi **106** offers the algorithm to the system for playing tic tac toe. The user interacts with the driver-vehicle interface **104** to decide who goes first, the human or robot arm **101**. This information is taken in by the raspberry pi **106** and if the human chooses to go first then the algorithm proceeds to begin the movement of the arm **101** after it recognizes the human's turn. If the human chooses to let the robot go first, then the algorithm activates the robot's turn. The algorithm then continues to allow the player to play against the robot arm in a competitive game of tic tac toe, where the robot arm **101** uses a grip method to physically pick-up and drop its pieces to the desired position in the game. Reed sensors

are used under the board to help identify which spaces are open and which are used. After each piece is placed, the camera **102** sees where the piece has been placed and signifies the owner of that space on the board **105**. When three of the same pieces have been recognized in a horizontal, diagonal, or vertical fashion, the camera **102** recognizes this, and the system denounces the winner. If players have filled the board with no recognition of three-in-a-row, the system calls a tie.

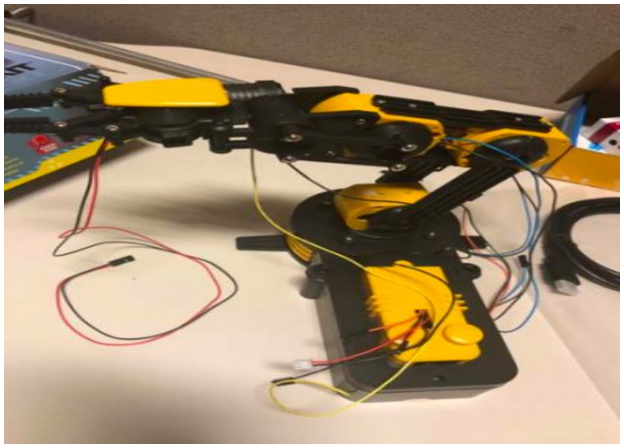
Project Implementation Plan

Terminator '19 Project Implementation Plan				
Task	Member	Start	Days	End
Project Implementation Plan design	Group	14-Jan-19	7	21-Jan-19
Bill of Materials	Charles/TK	22-Jan-19	4	25-Jan-19
Progress Assessment/Order Parts	Group	22-Jan-19	11	1-Feb-19
Completed Control Algorithm		28-Jan-19	42	11-Mar-19
Newell and Simon's algorithm	Owens/Marcus	4-Feb-19	5	8-Feb-19
Human Interface	Owens/Charles	11-Feb-19	5	15-Feb-19
Minimax algorithm	Owens/Marcus/Charles	18-Feb-19	5	22-Feb-19
Camera vision algorithm	Charles/Marcus	25-Feb-19	15	11-Mar-19
Power Supply	Marcus/Charles/TK	25-Feb-19	5	1-Mar-19
Camera Vision		25-Feb-19	25	22-Mar-19
Arduino camera setup	Marcus/Charles/Chibuike	25-Feb-19	15	11-Mar-19
Camera stand	Chibuike/TK	11-Mar-19	12	22-Mar-19
Board Hardware		18-Mar-19	12	29-Mar-19
Motors	TK/Chibuike/Marcus	18-Mar-19	12	29-Mar-19
Sensors	Charles/Chibuike/Owens	18-Mar-19	12	29-Mar-19
Integration	Group	1-Apr-19	5	5-Apr-19
Prototype	Group	5-Apr-19	8	12-Apr-19
Testing	Group	1-Apr-19	19	19-Apr-19

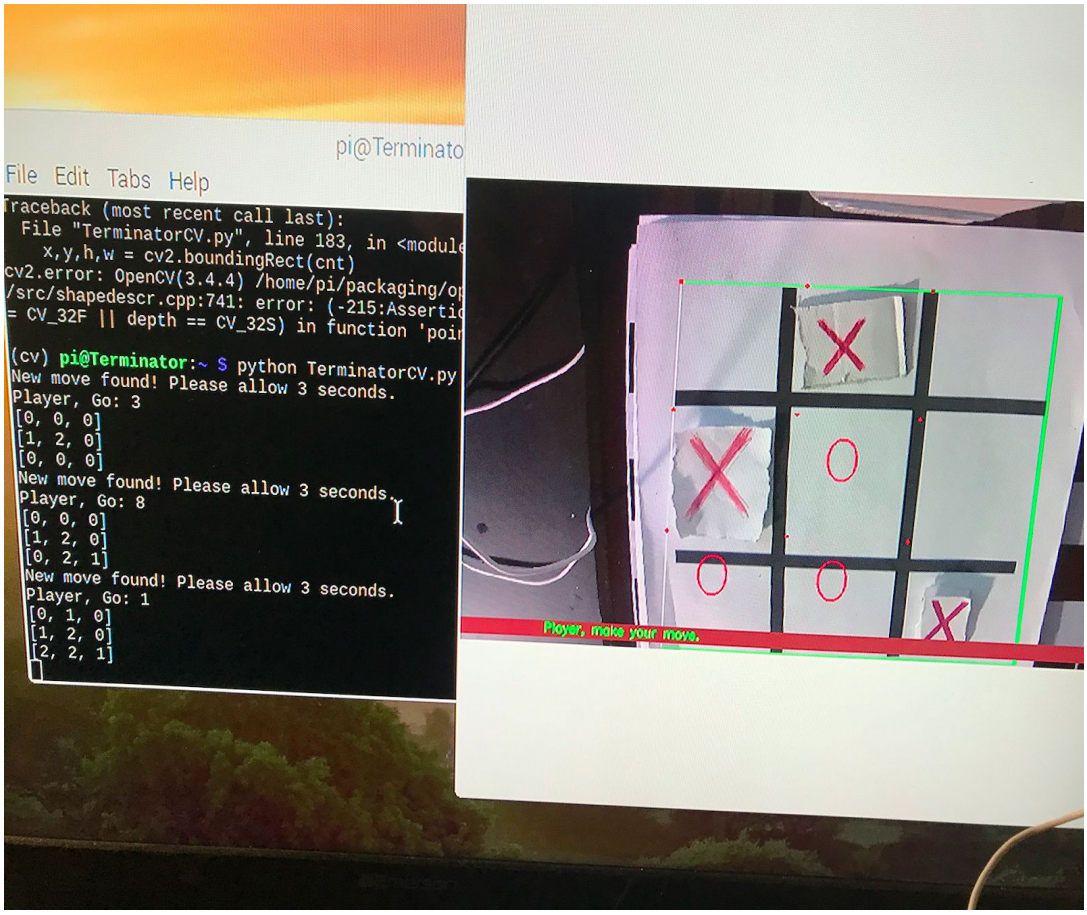
Project Implementation Progress

Team Terminator achieved the 2018-2019 academic year goal and then some. The Raspberry Pi camera is 100% capable of sensing the game board and correctly playing tic tac toe, while adapting to real in game situations.

Assembled Terminator Arm



Tic Tac Toe Algorithm Being Executed in Terminal



Python Tic Tac Toe Code

```
import operators as op
import random
from datetime import datetime

#create the game board
game = [[0,0,0],[0,0,0],[0,0,0]]
win = False

#this function writes in board the mark
def writePosition(position, player):
    if position < 3:
        if int(game[0][position]) is 0:
            game[0][position] = player
        else:
            print "Essa casa ja esta preenchida!"
            getPlayerMark(player)
    if position > 2 and position < 6:
        if int(game[1][position - 3]) is 0:
            game[1][position - 3] = player
        else:
            print "Essa casa ja esta preenchida!"
            getPlayerMark(player)
    if position > 5 and position < 9:
        if int(game[2][position - 6]) is 0:
            game[2][position - 6] = player
        else:
            print "Essa casa ja esta preenchida!"
            getPlayerMark(player)

#this function get the position mark of the player
def getPlayerMark(player):
    position = int(raw_input('Qual posicao escolhe? (0-8)'))
    print "Voce escolheu a posicao %d." % position
```

```
def getPlayerMark(player):
    position = int(raw_input('Qual posicao escolhe? (0-8)'))
    print "Voce escolheu a posicao %d." % position
    writePosition(position, player)
    print game[0]
    print game[1]
    print game[2]

def verifyWinner(player):
    index = 0
    for number in game[0]+game[1]+game[2]:
        if number == 0:
            index += 1
    if index == 0:
        return "EMPATE"
    #check lines
    for line in [0,1,2]:
        points = 0
        for house in game[line]:
            if int(house) == player:
                points += 1
            if points == 3:
                return True
    #check columns
    for column in [0, 1, 2]:
        points = 0
        for line in [0,1,2]:
            if int(game[line][column]) == player:
                points += 1
            if points == 3:
                return True
    #check diagonals
    points = 0
    for line in [0,1,2]:
```

```
        for line in [0,1,2]:
            if int(game[line][line]) == player:
                points += 1
            if points == 3:
                return True
        points = 0
        for line in [2,1,0]:
            if int(game[line][2-line]) == player:
                points += 1
            if points == 3:
                return True

        return False

def inteligence(player):
    objects = []

    #check lines, make a object for each line
    for line in [0,1,2]:
        blank = 0
        other = 0
        play = 0
        for column in [0,1,2]:
            field = int(game[line][column])
            if field is 0:
                blank += 1
            elif field is player:
                play += 1
            else:
                other += 1
        objects.append(op.elements(other, blank, play))

    #check columns, make a object for each column
```

Python Image Processing Code

```
import numpy as np
import cv2
import tictactoe as tic
import time

CARD_COLOR_UP = np.array([255,255,255],dtype="uint8")
CARD_COLOR_LOW = np.array([150,150,150],dtype="uint8")
FONTE = cv2.FONT_HERSHEY_SIMPLEX
global choose_user, player, computer
KERNEL = np.ones((3,3),np.uint8)
choose_user = True
game = True
init = False
last_time = 0

def findBiggestContour(mask):
    temp_bigger = []
    img1, cont, hier = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
    if len(cont) == 0:
        return False
    for cnt in cont:
        temp_bigger.append(cv2.contourArea(cnt))
    greatest = max(temp_bigger)
    index_big = temp_bigger.index(greatest)
    key = 0
    for cnt in cont:
        if key == index_big:
            return cnt
            break
        key += 1

def getField(coord, frame):
    crop_img = frame[coord[0]:coord[1], coord[2]:coord[3]]
    return crop_img


def getContoursHSV(img):
    mask = cv2.inRange(cv2.cvtColor(img, cv2.COLOR_BGR2HSV), CARD_COLOR_LOW, CARD_COLOR_UP)
    if len(mask) > 0:
        dilation = cv2.dilate(mask,KERNEL,iterations = 2)
        cnt = findBiggestContour(dilation)
        return cnt
    else:
        return False
```

```
    elif choose_user == False:
        tic.inteligence(computer)
        result = tic.verifyWinner(computer)
        print tic.game[0]
        print tic.game[1]
        print tic.game[2]
        if  result == True:
            print "O computador venceu!"
            break
        elif result == "EMPATE":
            print "Houve empate!"
            break
        write_comp = True
        choose_user = True

    if write_comp == True:
        if computer == 1:
            text = "X"
        else:
            text = "O"
        gameboard = np.array(tic.game[0]+tic.game[1]+tic.game[2])
        indexs = np.where(gameboard==computer)[0]
        for index in indexs:
            coord = (fields[index][2]+50, fields[index][0]+100)
            cv2.putText(frame, text,coord, FONTE, 3,(50,0,255),2,cv2.LINE_AA)


    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
    cv2.imshow('Exemplo',frame)

cap.release()
cv2.destroyAllWindows()
```

Conclusion

A significant amount of knowledge has been gained from this project. Specifically when it comes to the creation of a physical prototype. The ability to move from idea to functional prototype is invaluable to the thousands of companies based in the US. The project also gave experience on presenting/demos, which are required to drive an engineer's inventions.First of all, it helped us in managing a project and sharing the load of work. Secondly, it showed that simple but well adapted algorithms are often more efficient than more general and complex ones. Lastly, it gave us the opportunity to work at the interface between three related disciplines: Artificial Intelligence, Vision and Robotics, which lead to very interesting issues when studied together. This project is definitely one that can be continued and will be viewed as enjoyable by VIP participants.

References

1. A Simple Autonomous Robotic Manipulator for playing Chess against any opponent in Real Time
   http://www.nandanbanerjee.com/files/ICCVR-08AUG12-011%20paper.pdf
2. Deep Learning Machine Teaches Itself Chess in 72 Hours, Plays at International Master Level
   https://www.technologyreview.com/s/541276/deep-learning-machine-teaches-itself-chess-in-72- hours-plays-at-international-master/
3. Raspberry Turk
   http://www.raspberryturk.com/