WWW.MWFTR.COM

Team Terminator

A Tic-Tac-Toe Playing Robot

Cory Bethrant & Maxime Keita

Graduate Advisor: Chidi Ekeocha
Faculty Advisor: Dr. Charles Kim

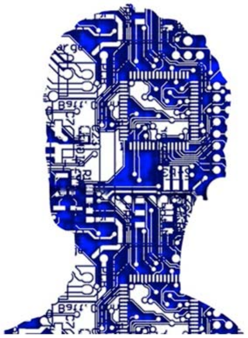
2nd EECS Day April 20, 2018

Electrical Engineering and Computer Science
Howard University

Problem Definition

Long Term Goal is to create an AI Robot capable of playing multiple games like Tic-Tac-Toe and Chess in the Physical Space for Cheap.

2017-18 Goal is to create an AI robot capable of playing Tic-Tac-Toe



Problem Statement

Is it possible to create an AI Robot capable of playing games in the physical space?

Design Requirements

Can Cost Be Kept Below $125 USD?

Follows all FCC Rules and Regulations as components are already thoroughly tested according to the FCC’s standards for Electronic Devices.

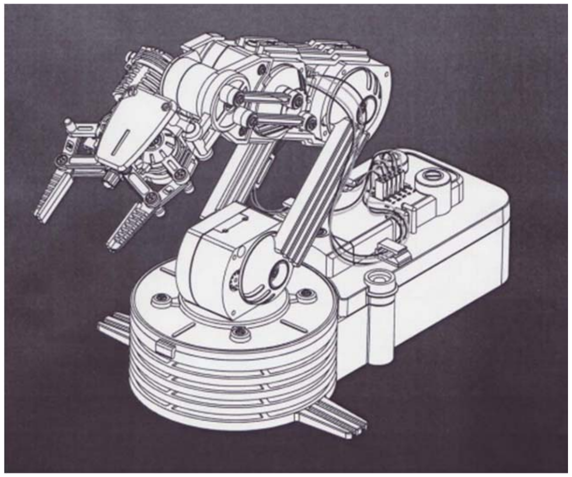
WARNING:
CHOKING HAZARD
Small parts. Not for
children under 3 yrs.

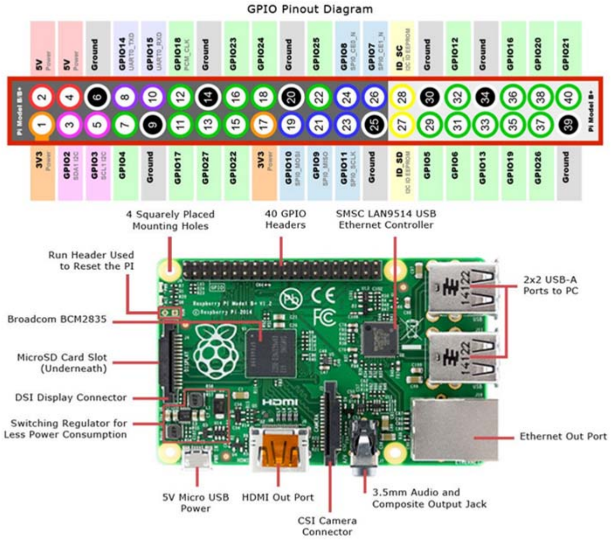
Current Status of Art

A paper based on this project was presented at the International Conference on Computer Vision and Robotics 2012 held at Bhubaneswar, India.

There is no machine learning component in this device. It uses a brute force algorithm to compete.

Solution Design

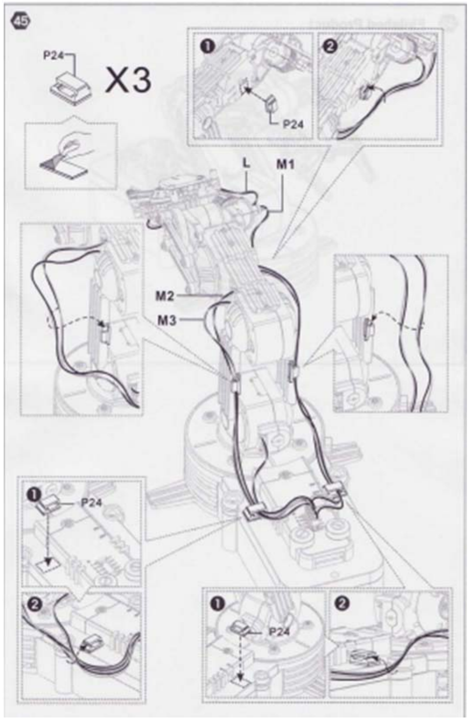




GPIO Pinout Diagram

Pin	Name	Pin	Name
1	3V3 Power	2	5V Power
3	GPIO2 SDA1 I2C	4	5V Power
5	GPIO3 SCL1 I2C	6	Ground
7	GPIO4	8	GPIO14 UART0_TXD
9	Ground	10	GPIO15 UART0_RXD
11	GPIO17	12	GPIO18 PCM_CLK
13	GPIO27	14	Ground
15	GPIO22	16	GPIO23
17	3V3 Power	18	GPIO24
19	GPIO10 SPI0_MOSI	20	Ground
21	GPIO9 SPI0_MISO	22	GPIO25
23	GPIO11 SPI0_SCLK	24	GPIO8 SPI0_CE0_N
25	Ground	26	GPIO7 SPI0_CE1_N
27	ID_SD I2C ID EEPROM	28	ID_SC I2C ID EEPROM
29	GPIO5	30	Ground
31	GPIO6	32	GPIO12
33	GPIO13	34	Ground
35	GPIO19	36	GPIO16
37	GPIO26	38	GPIO20
39	Ground	40	GPIO21

Pi Model B/B+
Pi Model B+
4 Squarely Placed Mounting Holes
40 GPIO Headers
SMSC LAN9514 USB Ethernet Controller
Run Header Used to Reset the PI
2x2 USB-A Ports to PC
Broadcom BCM2835
MicroSD Card Slot (Underneath)
DSI Display Connector
Switching Regulator for Less Power Consumption
Ethernet Out Port
5V Micro USB Power
HDMI Out Port
CSI Camera Connector
3.5mm Audio and Composite Output Jack



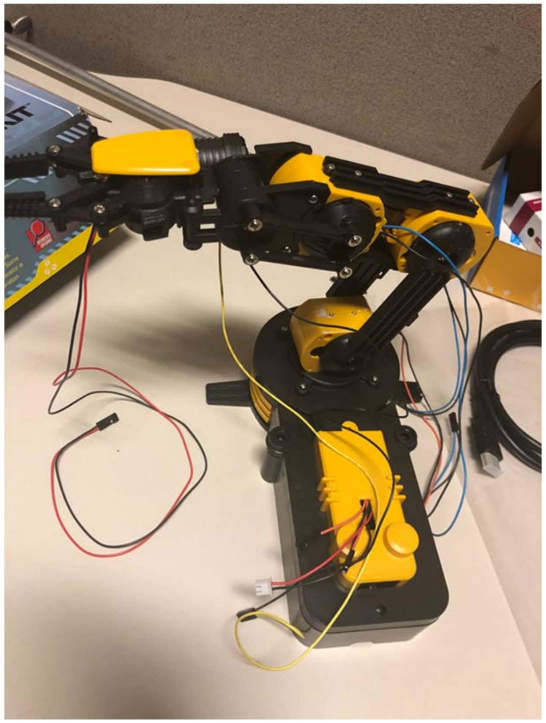
45
P24
X3
1
2
P24
L
M1
M2
M3
1
P24
2
1
P24
2

Implementation Process

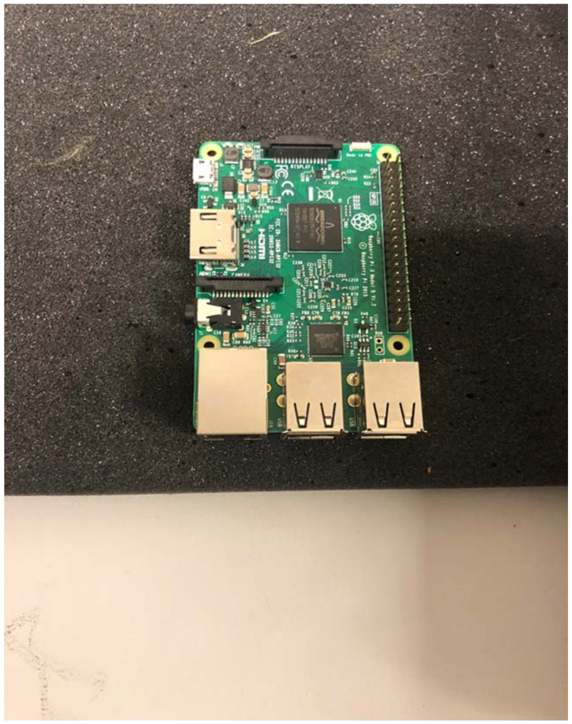
Major Steps

- Assemble Arm and Controller/Raspberry Pie and Mount Camera
- Develop MiniMax algorithm to use recursive machine learning to defeat the human opponent
- Use OpenCV to feed in real-time data to algorithm to determine where the board is and where pieces are located. (Pieces are required to be in frame)
- Modify MiniMax Algorithm to use the Arm and OpenCV events instead of console for game output

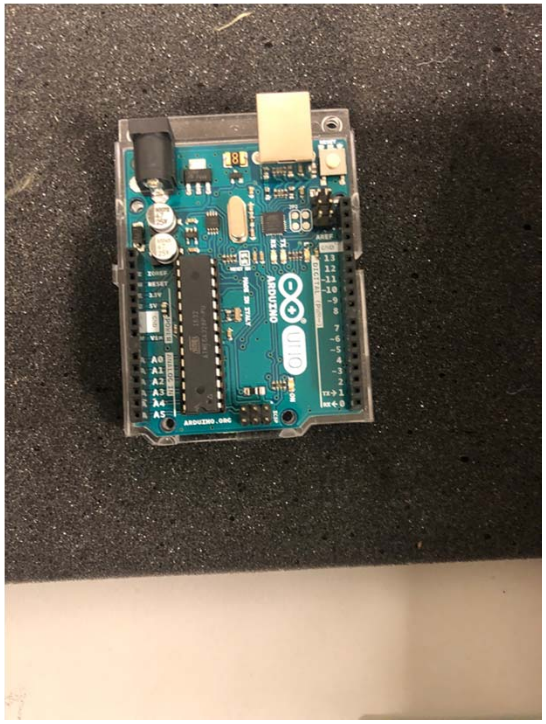
Arm Assembly



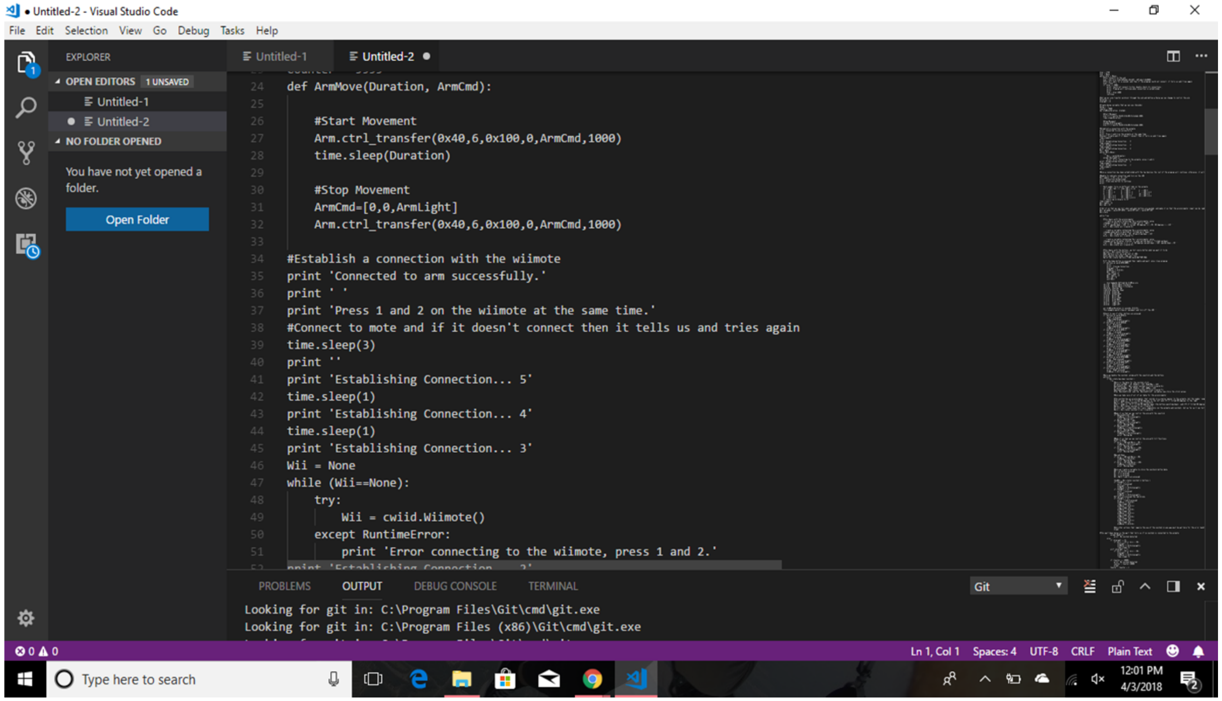
Raspberry Pi



Arduino Uno



Arm Movement



You have not yet opened a folder.

```
def ArmMove(Duration, ArmCmd):

    #Start Movement
    Arm.ctrl_transfer(0x40,6,0x100,0,ArmCmd,1000)
    time.sleep(Duration)

    #Stop Movement
    ArmCmd=[0,0,ArmLight]
    Arm.ctrl_transfer(0x40,6,0x100,0,ArmCmd,1000)

#Establish a connection with the wiimote
print 'Connected to arm successfully.'
print ' '
print 'Press 1 and 2 on the wiimote at the same time.'
#Connect to mote and if it doesn't connect then it tells us and tries again
time.sleep(3)
print ''
print 'Establishing Connection... 5'
time.sleep(1)
print 'Establishing Connection... 4'
time.sleep(1)
print 'Establishing Connection... 3'
Wii = None
while (Wii==None):
    try:
        Wii = cwiid.Wiimote()
    except RuntimeError:
        print 'Error connecting to the wiimote, press 1 and 2.'
```

```
Looking for git in: C:\Program Files\Git\cmd\git.exe
Looking for git in: C:\Program Files (x86)\Git\cmd\git.exe
```

MiniMax Algorithm

game.cpp

```
#include <iostream>
#include <sstream>
#include <iomanip>
#include "game.h"

using namespace std;

Game::Game() {
        for(int i = 0; i < 3; i++) {
                for(int j = 0; j < 3; j++) {

        board[i][j] = '-';
                }
        }
}

void Game::printBoard() {
        cout << "-------------------";
        for(int i = 0; i < 3; i++) {
                cout << '\n' << "|";
                for(int j = 0; j < 3; j++) {

        cout << setw(3) << board[i][j] << setw(3) << " |";
                }
        }
        cout << '\n' << "-------------------" << '\n';
}

bool Game::gameOver() {
        if(checkWin(HUMAN)) return true;
        else if(checkWin(AI)) return true;

        bool emptySpace = false;
        for(int i = 0; i < 3; i++) {
                if(board[i][0] == '-' ||
board[i][1] == '-' || board[i][2] == '-
```

```
')

        emptySpace = true;
        }
        return !emptySpace;
}

bool Game::checkWin(Player player) {
        char playerChar;
        if(player == HUMAN) playerChar = human;
        else playerChar = ai;

        for(int i = 0; i < 3; i++) {
                // Check horizontals
                if(board[i][0] == playerChar &&
board[i][1] == playerChar
                        &&
board[i][2] == playerChar)
                        return
true;

                // Check verticals
                if(board[0][i] == playerChar &&
board[1][i] == playerChar
                        &&
board[2][i] == playerChar)
                        return
true;
        }

        // Check diagonals
        if (board[0][0] == playerChar && board[1][1] == playerChar
                && board[2][2] == playerChar) {
                return true;
        } else if (board[0][2] == playerChar && board[1][1] ==
playerChar
                && board[2][0] == playerChar) {
                return true;
        }

        return false;
}
```

```
int Game::score() {
        if(checkWin(HUMAN)) { return 10; }
        else if(checkWin(AI)) { return -10; }
        return 0; // draw
}

Move Game::minimax(char AIboard[3][3]) {
        int bestMoveScore = 100; // -100 is arbitrary
        Move bestMove;

        for(int i = 0; i < 3; i++) {
                for(int j = 0; j < 3; j++) {

        if(AIboard[i][j] == '-') {

                AIboard[i][j] = ai;

                int tempMoveScore =
maxSearch(AIboard);

                if(tempMoveScore <=
bestMoveScore) {


        bestMoveScore = tempMoveScore;


        bestMove.x = i;


        bestMove.y = j;

                }

                AIboard[i][j] = '-';

        }

                }
        }

        return bestMove;
}
```

MiniMax Algorithm

Game.cpp - cont

```
int Game::maxSearch(char AIboard[3][3]) {
          if(gameOver()) return score();
          Move bestMove;

          int bestMoveScore = -1000;
          for(int i = 0; i < 3; i++) {
                    for(int j = 0; j < 3; j++) {
          if(AIboard[i][j] == '-') {
                    AIboard[i][j] = human;
                    int tempMoveScore =
minSearch(AIboard);
                    if(tempMoveScore >=
bestMoveScore) {
          bestMoveScore = tempMoveScore;
          bestMove.x = i;
          bestMove.y = j;
                    }
                    AIboard[i][j] = '-';
          }
                    }
          }
          return bestMoveScore;
}

int Game::minSearch(char AIboard[3][3]) {
          if(gameOver()) return score();
          Move bestMove;

          int bestMoveScore = 1000;
```

```
for(int i = 0; i < 3; i++) {
                    for(int j = 0; j < 3; j++) {
          if(AIboard[i][j] == '-') {
          AIboard[i][j] = ai;
          int tempMove = maxSearch(AIboard);
          if(tempMove <= bestMoveScore) {
                    bestMoveScore = tempMove;
                    bestMove.x = i;
                    bestMove.y = j;
          }
          AIboard[i][j] = '-';
                              }
                    }
          }
          return bestMoveScore;
}

void Game::getHumanMove() {
          int x, y = -1; // arbitrary assignment to init loop
          while(x < 0 || x > 2 || y < 0 || y > 2) {
                    // Loop until a valid move is entered
                    cout << "Enter your move in
coordinate form, ex: (1,3)." << endl;
                    cout << "Your Move: ";
                    char c;
                    string restofline;
                    cin >> c >> c;
                    x = c - '0' - 1;
                    cin >> c >> c;
                    y = c - '0' - 1;
                    getline(cin, restofline); // get
garbage chars after move
```

```
}
          board[x][y] = human;
}

void Game::play() {
          int turn = 0;
          printBoard();
          while(!checkWin(HUMAN) && !checkWin(AI) &&
!gameOver()) {
                    // human move
                    if(turn % 2 == 0) {
          getHumanMove();
          if(checkWin(HUMAN)) cout << "Human Player
Wins" << endl;
          turn++;
          printBoard();
                    } else {
          cout << endl << "Computer Player Move:" <<
endl;
          Move AImove = minimax(board);
          board[AImove.x][AImove.y] = ai;
          if(checkWin(AI)) cout << "Computer Player Wins"
<< endl;
          turn++;
          printBoard();
                    }
          }
}
```

MiniMax Algorithm

game.h

```
#include <iostream>

using namespace std;

const char human = 'X';
const char ai = 'O';

enum Player { HUMAN, AI };

struct Move {
                int x;
                int y;
};

class Game {
                char board[3][3];
public:
                Game();

                void printBoard();
                // Prints the board pretty-ly

                bool gameOver();
                // Returns true if a winner has been found or there are no empty spaces

                bool checkWin(Player player);
                // Checks for a win

                void play();
                // Primary game driver, loops through turn-by-turn until there's
                // a winner or full game board (draw)

                void getHumanMove();
                // Takes in values from the input stream and places them on the board
                // if valid.  Expects input in coordinate notation, ex (1,3)

int score();
                // Function to score game board states based on their outcome
                // Returns 10 for human win, -10 for AI win, 0 for draw

                Move minimax(char AIboard[3][3]);
                // Returns the best AI move's x, y coords via the minimax algorithm

                int minSearch(char AIboard[3][3]);
                // minimax helper fn for finding the next move for AI player, chooses the
                // move with the least possible score

                int maxSearch(char AIboard[3][3]);
                // minimax helper fn for finding the next move for human player, chooses
                // the move with the least possible score
};
```

MiniMax Algorithm

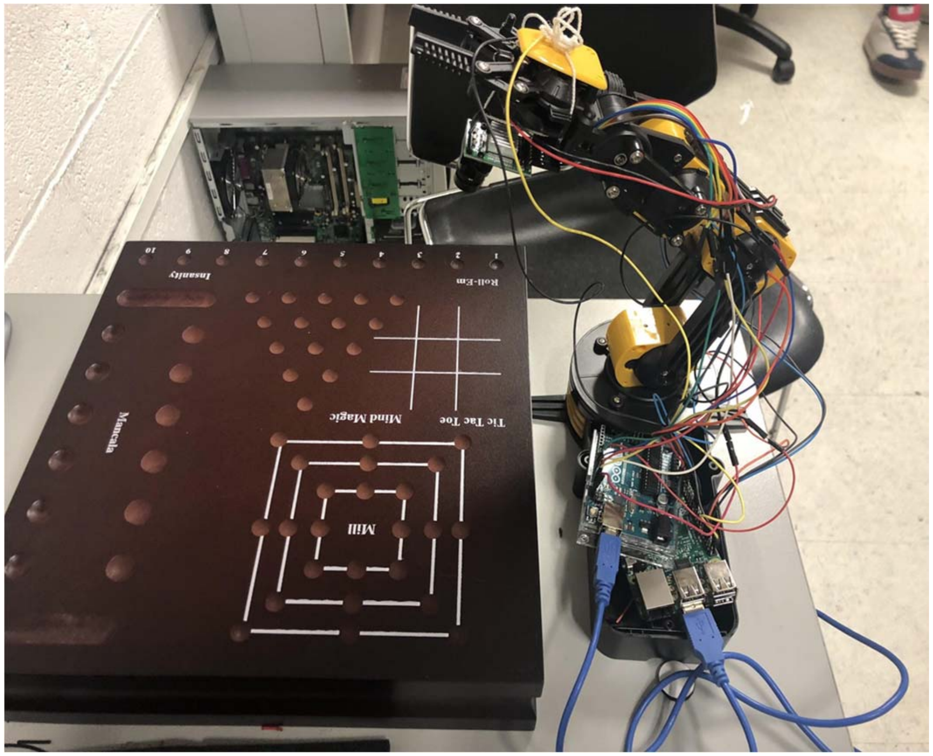
play.cpp

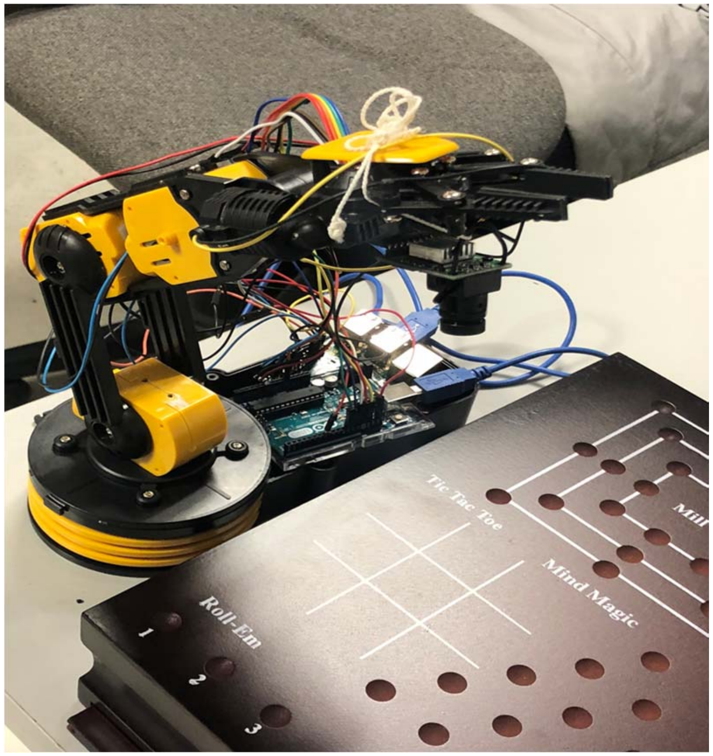
```cpp
#include <iostream>
#include "game.h"

using namespace std;

int main() {
		Game tictactoe;
		tictactoe.play();
		return 0;
}
```

Implementation





Conclusion

We have had the pleasure to see through the project in 8 months period of time.

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.

Acknowledgment

Special Thanks to the VIP Program for Making This Possible

VIP @Howard