RobBot

Sean Ficht

Robert Kriener

ECE/ARCH 868 Architectural Robotics

16 December 2011

YouTube Video: http://www.youtube.com/watch?v=QiUoS94ON1Q

Abstract

The RobBot is a personal assistant designed for use by hospital patients and/or in home care patients. Its purpose is to store and retrieve far away objects that a patient is incapable of accessing. The RobBot is able to recognize an item based on either the item itself or a picture of the item on a card. The patient can flash either the item or the card to the RobBot. The RobBot then recognizes which item it is supposed to retrieve/store and acts accordingly.

Scenario

Johnny B. Good went snowboarding and broke both his legs in a horrible accident. Johnny went to the doctor and found out that he would have to spend the majority of his time in his bed for the next couple of months. Rife with distraught, Johnny wondered what he would do. His doctor told him about a new invention called the RobBot. The doctor said it was a great way to be able to spend time in one place but still have access to all his personal items. Deciding to give it a try, Johnny rented a RobBot from the hospital. When he got home he installed the RobBot by giving it a correct mapping of his home. Mr. Good found that by simply showing the

RobBot a card of any object he desired that RobBot was then able to recognize the item, understand where it was stored, and retrieve that item for him. When he was done with the item, Johnny could show it to the RobBot. The RobBot would then retrieve the items appropriate drawer, return with the drawer, retrieve the item from him, and store it back away. Johnny decided that the RobBot was the most convenient thing he had ever had in his life. Thinking of the possibilities, Johnny B. Good decided to purchase a RobBot to use in even his regular everyday life.

Operation

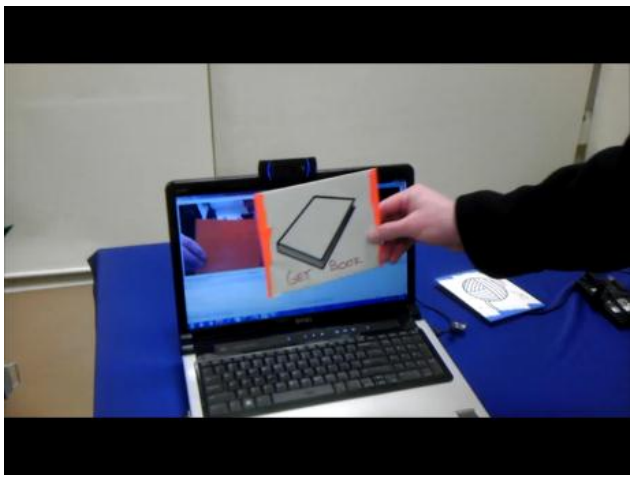
The RobBot has two primary purposes.

1. Retrieve an object based off a template of the object attached to a card.
2. Store an object based off seeing the object itself.

This is accomplished through three essential processes. The three main processes involved in the operation of RobBot are vision recognition, communication between the camera node and RobBot's controller, and motor control.

Vision Recognition

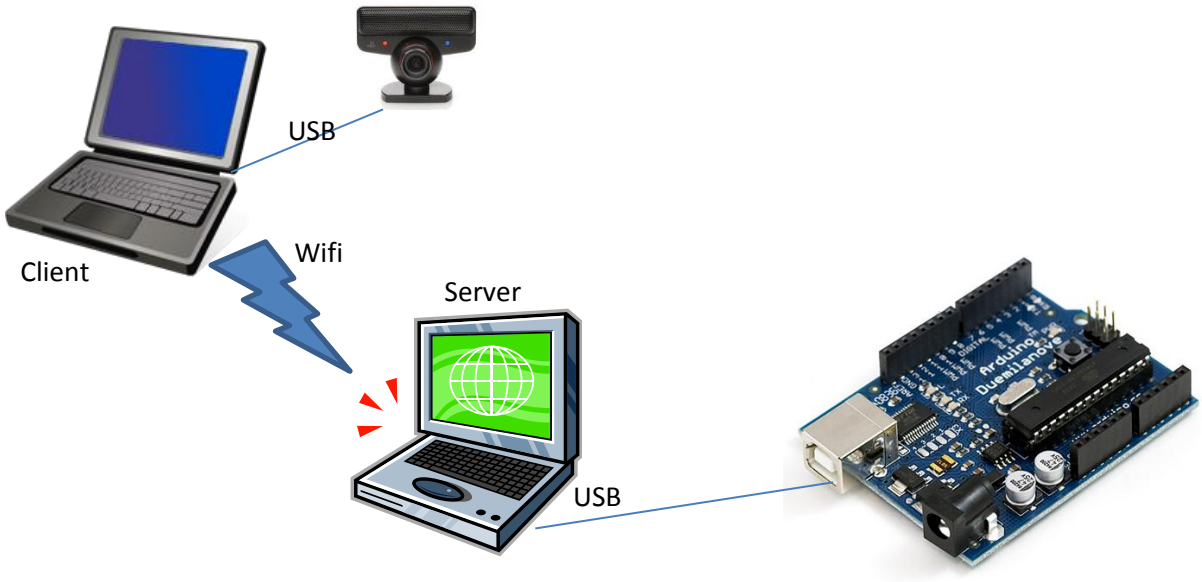
In its current form, the vision recognition system is based off of color coded cards as opposed to a template matching system.



Each color coded card corresponds to a particular item. When the card is shown to the camera, the vision algorithm determines which color it is being shown based on an RGB threshold. Once the color has been determined, the vision algorithm determines which object this color corresponds to, associates the object with the color, and then sends a message to the RobBot to retrieve that particular object.

Communication

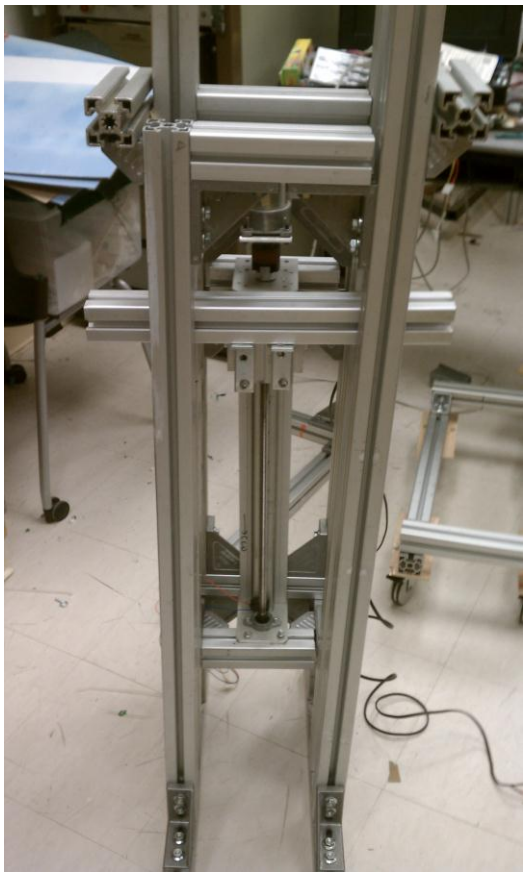
Once the vision algorithm has determined which object is to be retrieved, it then sends a message from the camera node to the RobBot control node via Wifi connection. The RobBot control node then signals the Arduino board, via serial communication, to move the arm position to the specified position.



Motor Control

On initialization, the Arduino instructs the motor to proceed downward until a limit switch is activated. Once this limit switch has been activated, the motor is instructed to move up about 20mm. The purpose of the limit switch is to establish a ground 0 value. Once the ground zero value has been established, RobBot is able to specify specific heights based on the ground 0 value. Hence, when receiving an instruction to retrieve a particular object the RobBot is told

what height to move its lift (arms) to and this position is at a positive height from ground 0. The motor control is performed using PID control but only using the Proportional part. The motor signal is generated via PWM where the pulse width determines the direction to turn and the speed at which to turn. Specifically a pulse of half a period corresponds to no motion while a zero pulse width corresponds to full speed rotation counterclockwise and a full pulse width corresponds to full speed rotation in the clockwise direction. The output PWM waveform is then subtracted from a half width PWM signal and sent to a linear amp and finally to the motor.



Code

```
// TemplateMatch_2.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "TemplateMatch_2.h"
#include "blepo.h"
```

```
#include "xPCUDPSock.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// The one and only application object

CWinApp theApp;

using namespace std;
using namespace blepo;

struct PackIn
{
	bool completed;
};

struct PackOut
{
	int action;
};

int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
{
	int nRetCode = 0;
	InitUDPLib();
	CUDPReceiver receiver(sizeof(PackIn), 30001);
	CUDPSender sender(sizeof(PackOut),30000,"172.22.113.59");

	// initialize MFC and print and error on failure
	if (!AfxWinInit(::GetModuleHandle(NULL), NULL, ::GetCommandLine(), 0))
	{
		// TODO: change error code to suit your needs
		_tprintf(_T("Fatal Error: MFC initialization failed\n"));
		nRetCode = 1;
	}
	else
	{

		//GOING TO HAVE TO COLOR CODE THE CARDS

		CaptureDirectShow cap;
		cap.BuildGraph(640,480,0);
		cap.Start();
		ImgBgr camera;
		Figure fig_camera;

		PackOut toSend;
		PackIn toReceive;

		int blue_count=0;
		int orange_count = 0;
		int green_count = 0;
		int yellow_count = 0;
		int red_count = 0;
```

```
            bool book_set = false, yarn_set = false, liftUp_set = false,
setDown_set = false, goHome_set = false;
            bool command_set = 0;
            toSend.action = 0;
            toReceive.completed = 0;



            while(1){
                  bool newimage = cap.GetLatestImage(&camera,1);
                  if(newimage){
                        fig_camera.Draw(camera);
                        for(int i=0;i<camera.Width();i++){
                              for(int j=0;j<camera.Height();j++){
                                    if((camera(i,j).b > 110 && camera(i,j).b
< 180) && (camera(i,j).g > 70 && camera(i,j).g < 110) && (camera(i,j).r > 40
&& camera(i,j).r < 80)){
                                          blue_count++;
                                    }
                                    if((camera(i,j).b < 80) && (camera(i,j).g
< 80) && (camera(i,j).r > 180)){
                                          orange_count++;
                                    }
                                    if((camera(i,j).b > 80 && camera(i,j).b <
120) && (camera(i,j).g > 100 && camera(i,j).g < 130) && (camera(i,j).r > 50
&& camera(i,j).r < 100)){
                                          green_count++;
                                    }
                                    if((camera(i,j).b > 45 && camera(i,j).b <
70) && (camera(i,j).g > 35 && camera(i,j).g < 75) && (camera(i,j).r > 120 &&
camera(i,j).r < 180)){
                                          red_count++;
                                    }
                                    if((camera(i,j).b > 50 && camera(i,j).b <
90) && (camera(i,j).g > 110 && camera(i,j).g < 160) && (camera(i,j).r > 130
&& camera(i,j).r < 180)){
                                          yellow_count++;
                                    }
                              }
                        }
                        //  1/4 the pixels
                        if(blue_count > 70000 && !command_set){
      //Was set at 76800
                              printf("Get Yarn\n");
                              toSend.action = 1;
                              //yarn_set = true;
                        }else if(orange_count > 70000 && !command_set){
                              printf("Get Book\n");
                              toSend.action = 2;
                              //book_set = true;
                        }else if(green_count > 70000 && !command_set){
                              printf("Lift Drawer\n");
                              toSend.action = 3;
                              //liftUp_set = true;
                        }else if(red_count > 70000 && !command_set){
                              printf("Set Down Drawer\n");
```

```
                              toSend.action = 4;
                              //setDown_set = true;
                        }else if(yellow_count > 70000 && !command_set){
                              printf("Return Home\n");
                              toSend.action = 5;
                              //goHome_set = true;
                        }else toSend.action = 0;
                        orange_count = 0;
                        blue_count = 0;
                        green_count = 0;
                        red_count = 0;
                        yellow_count = 0;

                        //if(toSend.action !=0) printf("Send: %d\n",
toSend.action);
                        sender.SendData(&toSend);
                        Sleep(1);
                        receiver.GetData(&toReceive);
                        cout << "Receive:" << toReceive.completed << endl;
                        if(toReceive.completed == 0){
                              command_set = 1;
                        }else command_set = 0;
                        //toReceive.completed = 1;


                  }
                  if(fig_camera.TestMouseClick()) break;
            }
      }

      return nRetCode;
}


// Project2.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <windows.h>
#include <mmsystem.h>
#include <stdlib.h>
#include "Serial.h"
#include "xPCUDPSock.h"
//#include <string.h>
#include <iostream>

#define LIFT_MIN 5
#define LIFT_MAX 350

using namespace std;

struct PackIn{
      int Action;
};

struct PackOut{
```

```
	bool Completed;
};

unsigned short UpdateTarget(const int &Action,const unsigned short &Target);

int _tmain(int argc, _TCHAR* argv[])
{
	_TCHAR Port[]=L"COM3";
	unsigned char Output[]="AAAZ"; //Package start byte is A and end is Z
	unsigned char Reply[5];
	int BytesWritten,BytesRead;
	unsigned short Target=0x4141;
	unsigned short NewTarget=0;
	PackIn UDPIn; //UDP I/O Vars
	PackOut UDPOut;
	int CurrentAction=0;

	//Open Serial To Arduino
	tstring commPortName(Port);
	Serial serial(commPortName,115200);

	// Initialize the UDP lib. If failed, quit running.
	if (!InitUDPLib()){
		cout << "Couldn't open UDP" << endl;
		return 2;
	}

	// Create receiver, with packet size equal to that of PACKIN and port
at 30000
	CUDPReceiver Receiver(sizeof(PackIn),30000);

	// Create sender, with packet size equal to that of PACKOUT and port at
30001,
	// and remote address 10.0.1.2(address of xPC target)
	CUDPSender Sender(sizeof(PackOut), 30001,"172.22.114.48");

	UDPOut.Completed=1;



	while(1){
		CurrentAction=0;

		// send data through sender
		//cout << "At location: " << UDPOut.Completed << endl;
		Sender.SendData(&UDPOut);
		Sleep(1);
		// get latest data from receiver
		Receiver.GetData(&UDPIn);
		Sleep(1);

		if((UDPIn.Action!=CurrentAction) && (UDPOut.Completed==1)){
//New command received and last was completed
			cout << "Received " << UDPIn.Action << endl;
			UDPOut.Completed=0;
			Sender.SendData(&UDPOut);
			CurrentAction=UDPIn.Action;
```

```
                  NewTarget=UpdateTarget(CurrentAction,Target);
                  if(NewTarget!=0){
                        Target=min(max(LIFT_MIN,NewTarget),LIFT_MAX);
                  }
                  bool CheckReceive=0;
                  while(!CheckReceive){
                        serial.flush();
                        Output[3]=90;//Z
                        Output[2]=Target;
                        Output[1]=Target>>8;
                        Output[0]=65;//A
                        BytesWritten = serial.write(Output,4);
                        cout << "Package" << Output << endl;
                        cout << BytesWritten << " bytes were written to the
serial port" << endl;
                        serial.flush();
                        Sleep(500);
                        BytesRead = serial.read(Reply, 3);
                        std::cout << Reply << "Was received, " << BytesRead
<< " bytes\n" << endl;

                        //Force receive check
                        CheckReceive=1;
                        for(int i=0;i<2;i++){

      CheckReceive=(Reply[i]==Output[i+1])&CheckReceive;
                        }
                  }
            }//End send new command

            //Check for command completed
            if(UDPOut.Completed==0){
                  BytesRead = serial.read(Reply, 2);
                  std::cout << Reply << "Was received, " << BytesRead << "
bytes\n" << endl;
                  Sleep(500);
                  if(Reply[0]==89 && BytesRead>0){//Y for yes
                        UDPOut.Completed=1;
                  }
            }
      }


      return 0;
}

unsigned short UpdateTarget(const int &Action,const unsigned short &Target){
      switch (Action){
            case 0://DO Nothing
                  return 0;
            case 1://Get yarn

      PlaySound(L"C://Users//Dysl3xik//Documents//ECE_868//Project2//CPPCode/
/Project2//Project2//Yarn.wav", 0, SND_FILENAME);
                  return 40;
            case 2://Get Book
```

```
      PlaySound(L"C://Users//Dysl3xik//Documents//ECE_868//Project2//CPPCode/
/Project2//Project2//Book.wav", 0, SND_FILENAME);
                  return 280;
            case 3://Lift Offset
                  return min(LIFT_MAX,Target+50);
            case 4://Setdown Offset
                  return max(LIFT_MIN,Target-25);
            case 5: //Home
                  return 10;
            default: //else fail
                  return 0;
      }
}
```

```
#define ACh 3

#define BCh 2

#define LimitSwitch 4

#define MotorVoltage 10

#define MotorBias 11

#define CountsPerRev 2500

#define RevsPermm 0.5


volatile int RevCount=0;

volatile int RevTarget=0;

volatile int Targetmm=0;

volatile int EncoderCount=0;


bool DoOnce=1;


int Temp;
```

```
int LoopCounter=0;


void setup(){
  pinMode(ACh, INPUT);
  pinMode(BCh,INPUT);
  digitalWrite(LimitSwitch,1);
  pinMode(LimitSwitch,INPUT);
  pinMode(MotorVoltage, OUTPUT);
  pinMode(MotorBias, OUTPUT);
  attachInterrupt(1, CheckEncoder, CHANGE);
  analogWrite(MotorBias,255/2);
  Serial.begin(115200);
}


void loop() {
  if(DoOnce){
     SetStartPoint();
     DoOnce=0;
  }
  //if(LoopCounter++%10==0){
    if(Serial.available()>=4){
       Temp=GetSerialTarget();
       if(Temp!=0){
         Targetmm=Temp;
```

```
        RunToTarget();
      }
    }
  //}
}

void CheckEncoder(){
  //A Goes Low
  if((digitalRead(ACh)==0)){
    if(digitalRead(BCh)==1){
      EncoderCount++;
    }
    else{
      EncoderCount--;
    }
  }

  //A Goes High
  else if(digitalRead(ACh)==1){
    if(digitalRead(BCh)==0){
      EncoderCount++;
    }
    else{
      EncoderCount--;
```

```
    }
  }

  if(EncoderCount>=CountsPerRev){
    RevCount++;
    EncoderCount=0;
  }
  else if(EncoderCount<=(-CountsPerRev)){
    RevCount--;
    EncoderCount=0;
  }

  RunToTarget();
}

int GetSerialTarget(){
  unsigned char beginbyte;
  unsigned char m1;
  unsigned char m2;
  uint8_t *IntPointer;
  unsigned char stopbyte;
  int target=0;

  beginbyte = Serial.read();
```

```
  while(beginbyte!=0x41){
    beginbyte = Serial.read();
  }
  //Serial.println("Synced");
  m1=Serial.read();
  m2=Serial.read();
  stopbyte=Serial.read();
  if (stopbyte ==0x5A){
    target=m1;
    target = target << 8;
    target = (target&0xFF00) | m2;
    Serial.write(m1);
    Serial.write(m2);
    //Serial.println(target);
  }

  if(target>0 && target<400){
    return target;
  }
  else{
    return 0;
  }
}
```

```
void RunToTarget(){
  RevTarget=ConvertToRevs(Targetmm);
  //Turn Positive Centered at half pwm 255/2->255 is turn positive
  if(RevTarget-RevCount>0){
    analogWrite(MotorVoltage, min( (abs(RevTarget-RevCount)*100+(255/2)) ,255) );

  }
  else{//Turn Negative centered at half pwm 0->255/2 is turn negative
    analogWrite(MotorVoltage, max( (255/2)-abs(RevTarget-RevCount)*100  ,0) );
  }

  if(RevTarget==RevCount){
    Serial.write("Y");
  }
}

double ConvertToRevs(int mmTarget){
  return (int)(RevsPermm*mmTarget*(-1)); //return negative because we switched direction of
motion
}

void SetStartPoint(){
  while(digitalRead(LimitSwitch)==1){
    //Serial.print("Im Here\n");
```

```
    Targetmm=-400;
    RunToTarget();
  }
  RevCount=0;
  Targetmm=100;
  RunToTarget();
}
```