Real-Time Image Processing in Mechatronics Systems – (RIMS)

Saman Keshmiri Amir Issaei Robert Willems

Project Sponsor:
Mr. Andrew Rushmere
Field Academic Coordinator
UBC Farm

Project 1152
Applied Science 479
Engineering Physics Project Laboratory
The University of British Columbia
January 9th, 2012

EXECUTIVE SUMMARY

Rodents have become a problem in a older greenhouse at the UBC Farm. They find the warm atmosphere of the greenhouse and the harvest and seedlings stored in the greenhouse a welcoming home. The rodents damage the hard work of the UBC Farm staff. Conventional rodent control mechanism have proven ineffective such as sonic traps which use high frequency ultrasound; snap traps require a great deal of attention and attract birds into the greenhouse when rodents are caught; resealing the greenhouse is far too costly; thus, the our team has designed an active device to track rodent movement within the bounds of the greenhouse and follow the rodents with the ability to trigger an external device for frightening the rodents. The external devices is out of the scope of our engineering team's knowledge and is left up to professionals in the field to implement. We present a real-time image-processing mechatronic system.

Using an infrared camera, to eliminate the distinction between daylight and night time, to capture images of the greenhouse and using image processing on the computer track rodents. The device installed in the greenhouse includes a housing mechanism for the camera with motors to control the position of the camera and consequently any external triggering device attached to it.

The tracking system is tested in the laboratory and has proven effective at tacking small objects in both regular light and low light conditions. There is a ~1.5s delay in motion present in front of the camera and reaction of the tracker; however, in the time frame present our team was not able to eliminate this delay completely but reduced it down from 2.8s to 1.5s using sophisticated probabilistic models for immediate future motion tracking.

The system has 5 components, 3 of which are coded on a computer using MATLAB computation software while the other 2 components are an Arduino microcontroller and the infrared camera. The computer and microcontroller communicate via a serial connection while information from the camera is read into the computer via Ethernet.

Accurate and precise motion detection allows the system to detect the smallest of changes in the environment it is monitoring thus making the system ideal for small rodents.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

This report is written to provide a complete and detailed description of the RIMS (Real-time Image-processing Mechatronics System) project for any team of engineers to replicate the prototype, modify, and make improvements to the system with great ease. RIMS is a rodent deterring system that uses image processing to track rodents in real-time in order to trigger an external rodent deterring device.

Background and Significance

RIMS is designed and prototyped for the UBC Farm for use in one of their greenhouses. During the fall and winter seasons the harvest is stored in the greenhouse while during the spring and summer seasons seedlings replace the harvest; both the harvest and seedlings are an ideal source of food for rodents when their natural food source outside the greenhouse depletes during the cold months of the year. The greenhouse is more than 20 years old; thus, the building's seals have started to deteriorate leaving convenient entry and exit routes for rodents. After a discussion with our project sponsor Andrew Rushmere, we are convinced our solution is a reasonable one to be discussed further in the *Methods* section of the report. The UBC Farm team has tested conventional rodent control methods, which have proven ineffective for this greenhouse. As well, perhaps the most obvious solution – cats – has proven ineffective, as coyotes in the area have been attacking the cats.

Conventional rodent control systems tested by the UBC Farm include snap traps and sonic traps. Sonic traps, which use high frequency ultrasound to irritate rodents and keep them out of the space, have proven ineffective as the rodents appear to become accustomed to the frequency after several days. Snap traps capture 2-3 rodents a week but they require constant maintenance as the traps can go off accidentally and must be replaced when a rodent is caught. Further, dead rodents caught in traps attract birds into the greenhouse, which introduces one more animal that finds the content of the greenhouse appetizing.

Other options, which have been considered but not tested, are light traps and complete resealing of the building. Light traps use a strobe light to frighten rodents from entering a space. Conventionally, they are placed near rodent nests; however, they are irritating to humans working in the greenhouse and they do not work during daylight hours. Moreover, they have not been tested but it could prove that rodents will get accustomed to these light traps as with the sonic traps. As well, their cost range is $60 - $200 each, which can be pricey since several are required to cover all entry points for the rodents. Finally, professional cost estimates for resealing the building are in the tens of thousands of dollars, too large of an investment for such an old building. Chemical rodent control methods cannot be used in this space due to their toxicity in an environment where food is handled. The chemicals can kill seedlings, and are harmful to humans if breathed in when humidity is high. Also, they are not sustainable and should not be washed down the drain as the case when the greenhouse is flooded to water the seedlings. Thus, chemical pest control is not an option.

Our solution attempts to be a cost effective alternative to the industry available rodent control systems that requires minimal attention and can be left to work indefinitely.

Project Objectives

The ultimate goal is to keep rodents from eating the harvest and seedlings stored in the greenhouse. RIMS is a programmable device to track rodents within the bounds of the greenhouse at all times and trigger an external device aimed at the rodent to frighten the rodent away from the harvest. One major goal of the system is for it to be robust and easily modifiable by any team of engineers to allow interchangeable rodent frightening systems as discussed in detail in the *future development* section of this report.

Scope and Limitations

The scope of this project is to engineer the RIMS device to detect and track rodent movement in the greenhouse. It is not in the scope of this project to determine an effective rodent frightening technology and research rodent behavior. Industry leading rodent control systems such as ultrasound and strobe lights are the source of the ideas for this project. We have selected a spotlight for frightening rodents as the best external device to test compared with the alternative ultrasound technology, which has proven ineffective from experience by the UBC Farm and is expensive to purchase high frequency sound generating circuits.

RIMS is designed with versatility in mind. The rodent detection system is robust and effective; the rodent deterring system is left for future implementation with long term field-testing. This is discussed later in the *recommendation* section of this report.

Several factors limit the variety of methods that can be implemented to achieve the goal of frightening rodents. The biggest challenge is the environmental conditions in the greenhouse; namely, temperatures as high as 50°C, extreme humidity and moisture year around, and water sprinkling from the ceiling to water plants. The device must be able to survive in such an environment. Further, human activity is high in the greenhouse – several hours a day with 2-4 people working in the greenhouse – thus the system must not irritate humans, pose a threat such as electric shock, and interfere with human activities. Finally, it must require minimal maintenance and attention; one of the major reasons the farm would like to stop using snap traps is the huge maintenance burden.

Organization

The remainder of this report explains the methods implemented in RIMS as part of the *Discussion* section as well as make recommendations for future development of the system and improvement. An important aspect of this project is the report to be used for future improvements of the project. The deterring system is not implemented due to the fact that budget constraints, time constraints, and uncertainty of rodent behavior persuaded our team that a tracking system free to have any deterring system attached is the best option to complete for this project. Details of software code, design CAD drawings, circuitry, and testing result calculations can be found in the appendices at the end of the report.

DISCUSSION

The discussion section of this report aims to inform the reader of the engineering process taken by our team in developing RIMS. The theory section attempts to communicate the reasons behind the methods used in RIMS. The methods/Test Protocol section provides details of the workings of RIMS.

Theory

This section of the report attempts to give the reader perspective for our decision to design a rodent tracking system leaving the deterring component to professionals of rodent behavior and habit. With the objective to deter rodents from eating the harvest in the greenhouse in question, our team initially thought of several mechanisms that could achieve this goal such as mechanical restrictions to stop rodents from accessing the food. After several discussions with out project sponsor, it became clear that an active device should be used; one which only reacts to rodents near it to eliminate the risk of rodents becoming accustomed to it. This design idea escalated to a mechatronics system which fires small objects at detected rodents. It is clear that all active devices require a detection system to operate. This is what inspired us to design the rodent tracking system – RIMS.

Research into the subject of rodent detection resulted in several technologies being considered with our final decision being an infrared camera and image processing. Other alternatives considered are infrared sensors, sonic sensors, and thermal imaging cameras. Each one of these methods have their own advantages and disadvantages as outlined in **Table 1**.

Table 1	Advantage	Disadvantage
Infrared Sensors	-Cost effective -Simple: when object is in range sensor sends signal to processing unit	-Very short range ~20-30cm -Long range sensors become expensive (>$30 each) and still no more than 1m range -No information of distance and direction
Sonic Sensors	-Information of distance and position of object	-Costly ~$15 per sensor -Short range ~1m -Sensitive to humidity and environment

Thermal Imaging Camera	-Does not require motion – can detect rodent from body heat -Easily distinguish human from rodent	-Very costly (>$500)

RIMS uses a thermal camera for night surveillance as its vision. A camera eliminates the need to have multiple sensors as it can view the entire space in the greenhouse. Further, the camera can be place on the ceiling of the greenhouse eliminating the need for sealed water resistant robotics at ground level. Further, the infrared allows the computer to view the space with any amount of light as the camera has infrared LEDs to light up the room during low light hours. The camera is a *Vansview infrared* camera.

Methods / Prototype

RIMS is a mechatronic system consisting of in infrared camera, 3 servo motors, an Arduino microcontroller, and steel casing to house the components and provide 2 axis of freedom for rotation. **Figure 1** is a picture of the assembled mechanical design.

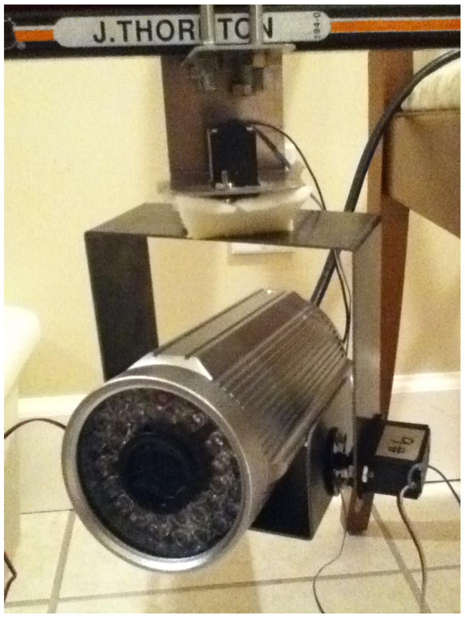

Figure 1 - Assembled prototype with servos, camera, and mount.

As seen in **Figure 1** the mechanism is cut from 1.6mm steel. Steel is not the ideal material of choice for the environment of our greenhouse; however, due to time constraints this was the only material available to us. Future prototypes, should there be further development, can take advantage of material better suited for hot and humid environments such as aluminum.

There is a piece of foam on top of the mounting structure. This is because the servo motor responsible for rotating the structure oscillates when rotating the device due to the large mass of the camera causing an over shoot of the desired angle. The foam acts as a damper in slowing down rotational movement and preventing the servo from over rotating due to inertia.

There is one servo which provides 360° rotation around a vertical axis, as well as two servomotors that provide the tilt ability of the device. With 2 axis of freedom, the camera field of view can capture the entire greenhouse from the center of the ceiling.

Flow Diagram / Algorithm

The following **Figure 2** displays the algorithm of the entire detection system. Captured images from the camera are used as the input to the system of the location we would like the system to point – namely towards the direction of detected motion where a rodent maybe. The images are also used as a feedback system for the device to determine how close it is to its desired location.

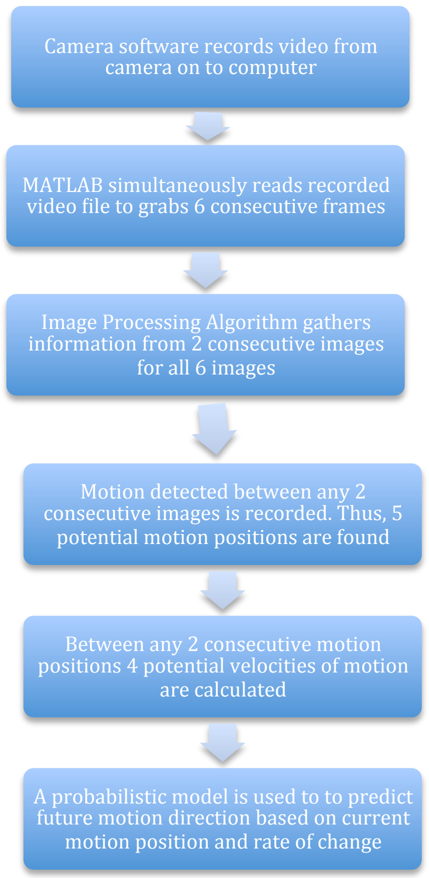
Camera software records video from camera on to computer
MATLAB simultaneously reads recorded video file to grabs 6 consecutive frames
Image Processing Algorithm gathers information from 2 consecutive images for all 6 images
Motion detected between any 2 consecutive images is recorded. Thus, 5 potential motion positions are found
Between any 2 consecutive motion positions 4 potential velocities of motion are calculated
A probabilistic model is used to to predict future motion direction based on current motion position and rate of change

Figure 2 – Algorithm

As seen in **Figure 2** the algorithm takes 6 consecutive images from the stream, detecting motion between any 2 consecutive images. A center of motion is calculated and between any two consecutive movements a velocity of motion is determined. This is used in the probabilistic model to predict future motion positions and rotate the camera in that direction.

The concept is similar to a simple PID control system with the exception that the desired location is a continuous function of time constantly moving and a simple PID control system creates a large lag in movement. The predictive nature of the control system allows for rodent tracking with high latency camera outputs such as our camera. Detailed explanations of each of the 5 components of the system are described in detail with **Figure 3**.

The camera manufacturer has provided a CD with software that allows for recording of video to file form the camera. This software is the best mechanism to access images from the camera as it has the least latency of ~2.8s. The camera is equipped with an Ethernet output where the video stream can be read through RTSP – real-time streaming protocol. This particular camera outputs a stream that MATLAB cannot read using the RTSP protocol. Thus our only option is to record video to disk and read it into MATLAB. Nevertheless, MATLAB has the ability to read a video file as the file is being written; thus, there is no delay in recording while reading the file into MATLAB.

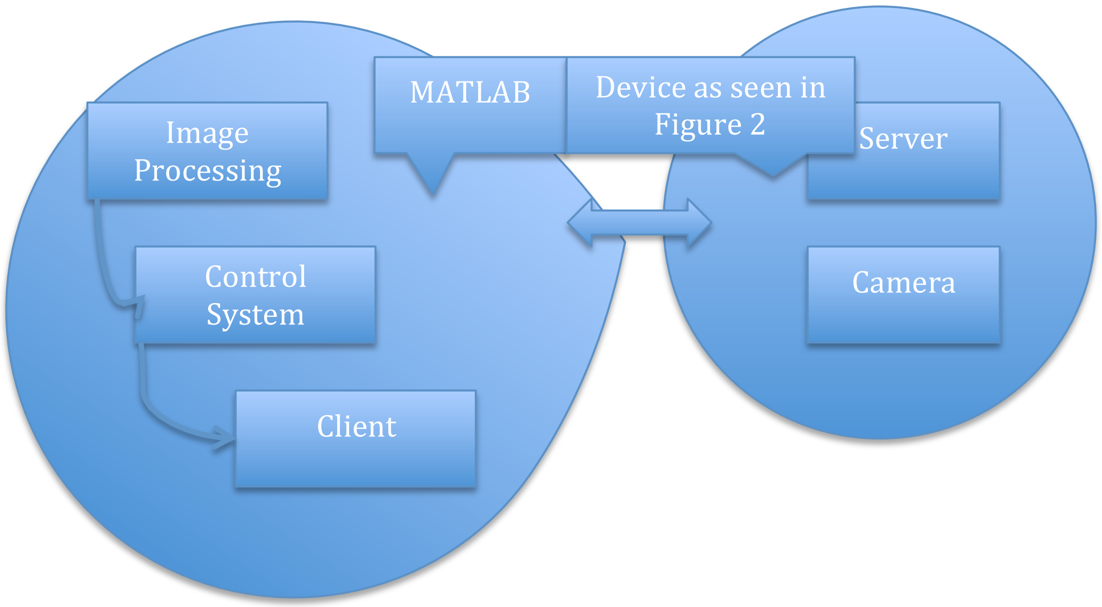

Figure 3 - System Components

The 5 components of the system are: Image processing, Control system, client, server, and the camera. The first 3 are written in MATLAB on a computer with the latter 2 as the device to be deployed in the field. Currently the connection between the two sides is through a virtual serial port using a USB cable.

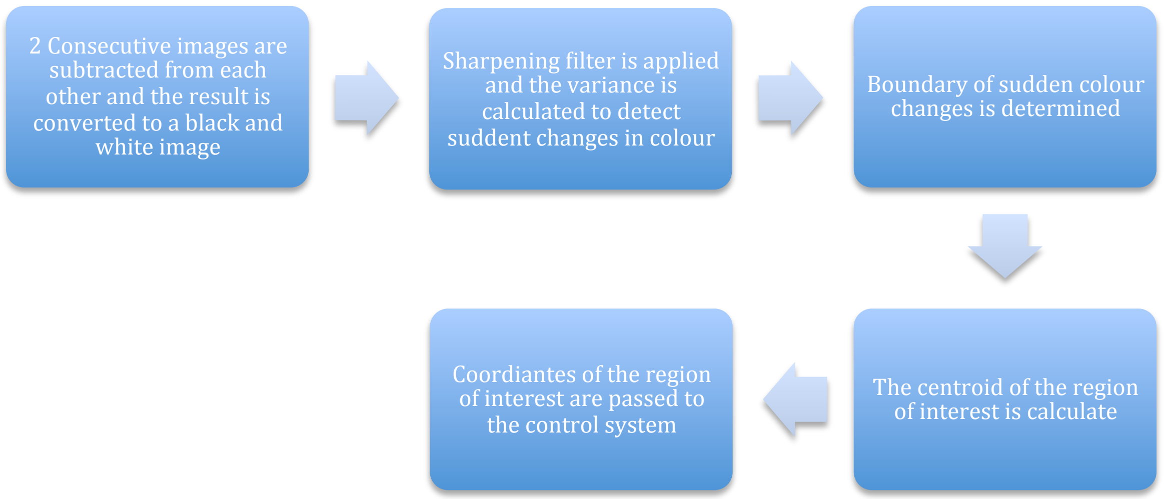
2 Consecutive images are subtracted from each other and the result is converted to a black and white image
Sharpening filter is applied and the variance is calculated to detect suddent changes in colour
Boundary of sudden colour changes is determined
The centroid of the region of interest is calculate
Coordiantes of the region of interest are passed to the control system

Figure 4 - Image Processing Algorithm

The Image processing component's algorithm is visualized in **Figure 4**. For any 2 consecutive images, the difference is found and converted into a black and white image. The conversion into black and white merges a 3 layer image into 1 layer thus processing performance is improved at least 2x. This difference image is put through a filter that sharpens abrupt colour changes and fed to a function that determines boundaries within an image. A boundary is defined as any place where sudden colour changes in adjacent pixels are found using the variance of the image. These boundaries are the regions of interest to us; namely, the area of motion. From the pictures below it can be seen that if no motion is present and two consecutive images are subtracted, the result is a blank image and nothing is done by the algorithm.

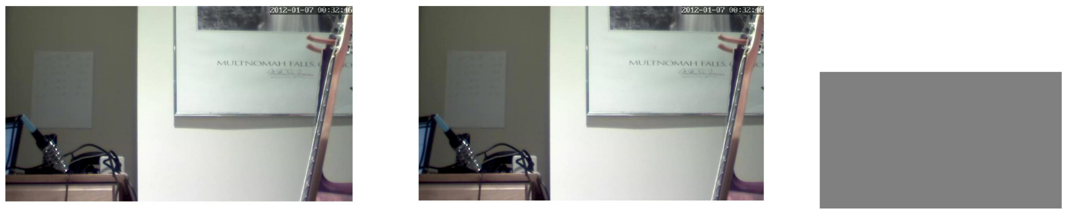

Figure 5 - No motion image subtraction

However, in the case that there is a motion detected the results are show in the images below.

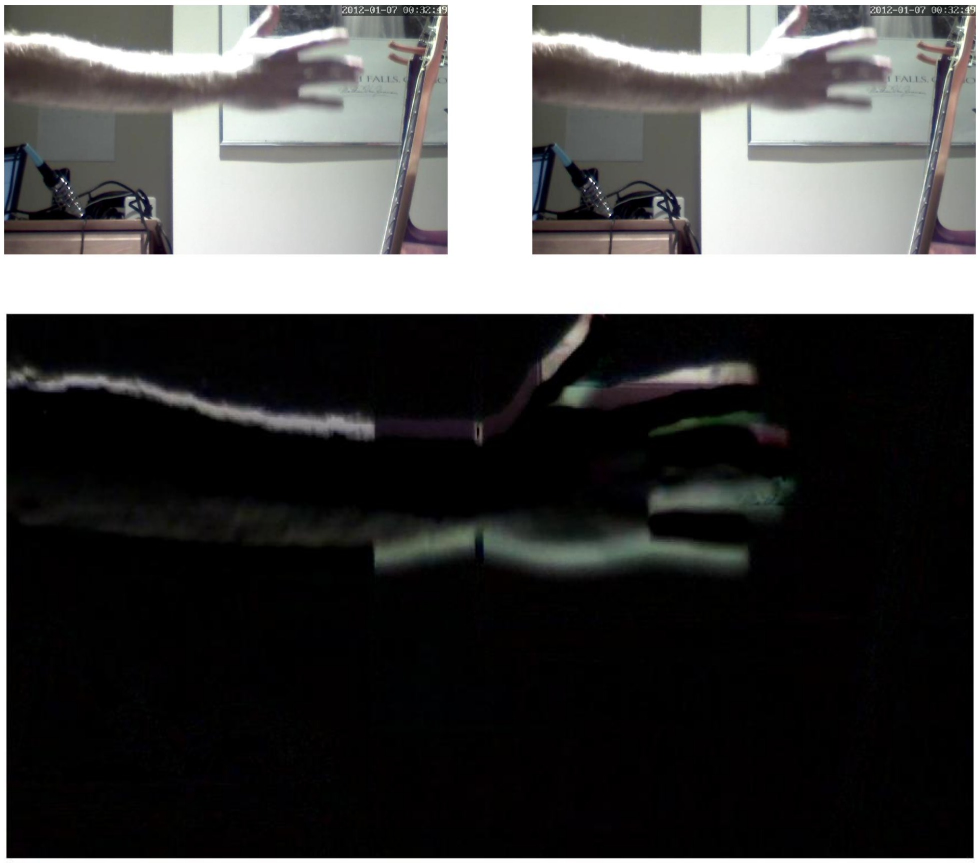

Figure 6 - Slight motion detection

It is clear that the subtraction of two consecutive images is very precise as hardly a difference to the human eye in the two above pictures is well established by the computer. Finally, the filters applied to the subtracted image and boundaries detected results in the following image:

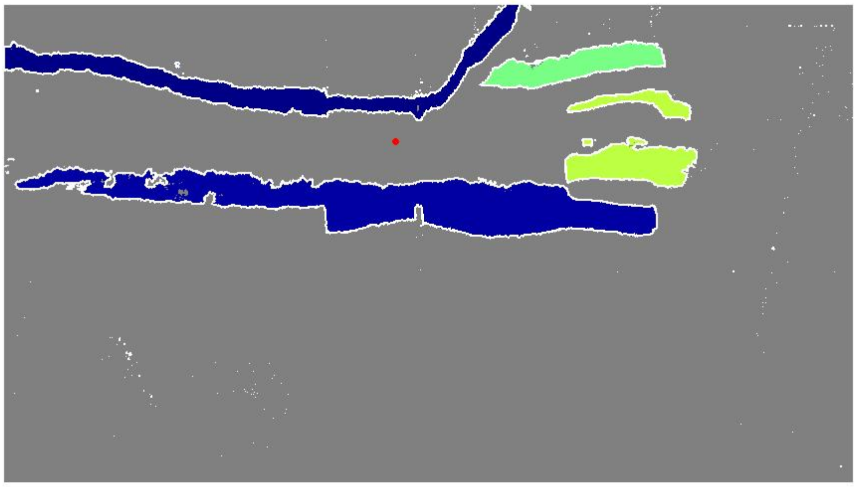

Figure 7 - Boundary detected for motion and region of interest outlined in white

Once the region of interest is detected, the centroid is calculated and the coordinates are passed to the control system.

The control system records 5 coordinates of the centroid for 5 separate regions of interest of 6 consecutive images. Then calculates the velocity of motion from the 5 centroids. Using this information predicts the immediate future's most probabilistic direction and begins a movement in that direction. The desired position of the camera is mapped to servo angles and this information is passed to the client in MATLAB to transmit this information to the server – the Arduino microcontroller.

The client-server relationship of MATLAB and Arduino microcontroller is though a USB connection acting as a virtual serial port. The server software on the Arduino microcontroller listens for serial requests and has built in function that can be called through MATLAB. The code on both sides of the system can be found in **Appendix A**.

Alternative Designs Considered

A great deal of time was devoted to brainstorming ideas for rodent deterring solutions before it was decided to design a tracking mechanism. This section expresses these ideas, which maybe useful for future groups working on this project.

The simplest designs considerations were given to stationary devices: a device called the ball-n-chain. It consists of a attachment ring for a table leg and they would be installed on every table leg. The device has a mass attached to the ring with a metal wire. Using a simple infrared motion sensor, the mass is begins spinning around the table leg when a rodent is detected trying to climb up the table. There are several major limitations with this design. First, many must be produced to solve the problem as there are many tables in the greenhouse where the harvest is stored on. The circuitry for the device must be hidden and protected from pouring water. As well, power delivery to the devices is a major issue. Wiring them through out the room is not practical and battery power requires management and replacement. A passive device to stop rodent from climbing up a table leg was considered as well but decided that rodents would get around it; thus, an active device must be engineered.

Another design was for a sweeping robotic arm installed on the tables. This robotic arm would sweep the tables at a desired elevation above the table when no humans are present in the greenhouse. The limitation for this idea was that, the device must be turned on and off by staff when they enter and exit the greenhouse. Also, setting the height above the tables would be problematic as seedlings could be damaged.

The most prominent idea and the runner up was the shooting robot. A device mobile on the greenhouse floor, roaming around and shooting a small arrow at a rodent when one is detected while it chases the rodent. This design went through several brainstorming sessions before it was abandoned. Power management was a major limitation. Battery power would require far too much management. It was proposed to use an magnetic induction charging technology where the robot would guide itself to a home base when it required a recharge. A second limitation was differentiating human feet from rodents and eliminating human contact. To engineer around both these limitations, the robot would be placed on rails installed on the greenhouse floor. The rails would serve to limit the robots movement around the room for human interference, and provide power. Our project sponsor informed us that the tables in the greenhouse are non-stationary and the rails can be problematic. Also, the material limitation of the rails is an issue as they must withstand constant water, mud and dirt, as well as people stepping on them all while providing adequate contact for power delivery.

It was from these ideas that we decided to only design and build a tracking mechanism, as all the above ideas require rodent tracking.

Results and Discussion of Results

The results of in lab testing of the device have revealed a great deal. The images gathered from the camera are 1280x720 pixels each. Each image is approximately 300 kb in size and there does not appear to be a limitation to the data transfer rates between the camera and the computer. The camera has onboard circuitry to decide when enough light is present to use regular frequency images and when to turn on its night mode and capture images in infrared. The device operates at 30 frames per second but only 6 consecutive frames are used at a time and this is due to the 2.8s latency present in the camera to deliver an image via Ethernet or component cables. This delay is due to the conversion and encoding of the images to a .sja formatted stream onboard the camera. This 2.8s lag in information transfer from the camera to the computer presented a major issue at first that required immediate attention. It is critical for the system to have minimal lag to detect rodents and point to them before they are gone. Our algorithm attempts to minimize this latency by using a probabilistic model to predict movement. This has allowed the lag of the system to decrease to ~1.5s even though information is still 2.8s delayed; however, every few seconds the system pauses for an instance to skip queued frames to the latest information available.

CONCLUSION

In conclusion, RIMS is fully operational and tested. It is able to track movement using the infrared camera with a ~1.5s delay. This delay is reduced to 1.5s from 2.8s; however, it is clearly present when testing the device. Infield testing may prove that this delay is adequate for the purpose of deterring rodents or that improvements must be made.

Within the time frame of our project, the rodent tracking system is must use the camera's manufacturer software to record video which is read by MATLAB simultaneously as it is recorded. Using the camera's software, a recording schedule of 1 hour intervals was setup so that there is no disruption to the stream of video into the processing software and as well disk space is reused as each previous recorded file is deleted.

It remains that the system be installed in the greenhouse for infield testing and an appropriate external triggering device such as a laser or strobe light be installed on the device to frighten rodents away.

The detail of motion detection is adequate in both daylight and low light conditions and the precision of the software can detect the smallest of changes as seen in **Figure 6**. The software easily detects motion between two consecutive images that cannot be seen by the human eye. Infact, the smaller the object in moving the better it is detected because the algorithm can find abrupt changes in the difference image much nicer when they are only a small section of the image and not covering the entire image.

PROJECT DELIVERABLES

We are presenting to our project sponsor and the Engineering Physics Project Laboratory the assembled mechanical design of our rodent tracking system as seen in **Figure 1**. The Arduino microcontroller is configured with the server site software require to communicate with MATLAB. Moreover, the MATLAB code is also included for the device to be operation. Setup of the device in the greenhouse is to be complete still but will be done by our team if requested by our project sponsor although a spare computer must be placed in the greenhouse as well to make the device operational. This report is also included to the project sponsor for passing down to future engineering teams to modify RIMS.

The camera used is owned by the Engineering Physics Project Laboratory and maybe requested to be returned by the lab. In this case, from this report the sponsor can purchase a camera as a permanent replacement.

RECOMMENDATIONS

RIMS is complete but since rodent behavior is highly unpredictable – to our project team at least – this project has great potential to be modified and expanded upon with great ease. The tracking system detects rodents and points the system at the rodent with ability to follow a moving rodent. The system can be modified to have a spotlight/strobe light, pressurized water gun, heat laser, or air pressure fire at the detected rodent. This way, after long term infield experimentation which ever method works best can be implemented.

Further, in order for the system to better track rodents, it uses a probabilistic model to predict and detected a rodent's next motion when following it. Future development of the project can be focused on improving this feature to better track rodents perhaps at higher speeds. The purpose of this feature is to reduce the latency present in the camera from when the image is captured to when the computer analyzes the images. With a better camera where latency is not an issue, this feature can be expanded upon to create a faster tracking mechanism for rodents moving at high speeds.

The setup of the entire system requires a computer running MATLAB to operate. Future development can be focused on using the Arduino Ethernet Shield along with the camera's Ethernet capabilities to completely separate the processing done on a remote computer and the device itself. The exact same server-client communication we are using over a virtual serial port with a USB connection can be replaced with a network using Ethernet. Some work on this was attempted during the design process; however, due to time restriction was not implemented.

APPENDIX A

MATLAB Code:

Camera Controller

```
classdef CameraController < handle
  properties(SetAccess = private, GetAccess = public)
     comPort = 'COM4'; %com port
     ard; %arduino connection object
     %pan motor constants
     pMotorPin = 2;
     pMotorCal = 0.015;
     pMotorPos = 55;
     pMaxAngle = 180;
     pMinAngle = 0;
     %tilt motor constants
     tMotorPin = 1;
     tMotorCal = 0.015;
     tMotorPos = 25;
     tMaxAngle = 180;
     tMinAngle = 0;
  end
  methods
     function obj = CameraController()
        obj.ard = arduino(obj.comPort);
        setup(obj);
     end %constructor
     function setup(a)
        a.ard.servoAttach(a.pMotorPin);
        a.ard.servoAttach(a.tMotorPin);
        servoWrite(a, a.pMotorPin, a.pMotorPos);
        servoWrite(a, a.tMotorPin, a.tMotorPos);
     end %setup
     function servoWrite(a, pin, angle)
        a.ard.servoWrite(pin, angle);
     end
     function moveCamera(a, x, y)
         display(sprintf('Rotate Servos - x = %d, y = %d',x, y));
         moveCameraX(a,x);
         moveCameraY(a,y);
     end %moveCamera
     function moveCameraX(a, x)
        if ~isnan(x)
           %change tilt motor position by y and write that to the motor
```

```
            a.pMotorPos = floor(a.pMotorPos + x*a.pMotorCal);
            if a.pMotorPos < a.pMinAngle
               a.pMotorPos = a.pMinAngle;
            elseif a.pMotorPos > a.pMaxAngle
               a.pMotorPos = a.pMaxAngle;
            end
            servoWrite(a, a.pMotorPin, a.pMotorPos);
         end
      end %moveCameraX
      function moveCameraY(a, y)
         if ~isnan(y)
            %change tilt motor position by y and write that to the motor
            a.tMotorPos = floor(a.tMotorPos - y*a.tMotorCal);
            if a.tMotorPos < a.tMinAngle
               a.tMotorPos = a.tMinAngle;
            elseif a.tMotorPos > a.tMaxAngle
               a.tMotorPos = a.tMaxAngle;
            end
            servoWrite(a, a.tMotorPin, a.tMotorPos);
         end
      end %moveCameraY
      %called before an object of the class is destroyed
      function delete(a)
         clear a.ard);
      end %delete
   end %methods
end %class
```

Tracking

```
classdef Tracking < handle
   properties(SetAccess = private, GetAccess = public)
      cameraController;  %camera pan and tilt controller
      directory = 'C:\Users\Rob Willems\Documents\Record\IPCamera\';
      videoAccessDuration = 0.2;
      currentFileName; %current complete video file name being accessed
      currentFileRecordStartTime;
      currentFrameStartTime;
      newRecordingStarted = false;
   end %properties
   methods
      function o = Tracking()
         o.cameraController = CameraController();
         o.currentFileName = getNewestFileName(o);
```

```
            display('Waiting for new recording...');
            while ~o.newRecordingStarted
                %wait until recording starts to process videos
                checkFileName(o);
            end
            o.newRecordingStarted = false;
            while true
                o.currentFrameStartTime = toc(o.currentFileRecordStartTime);
                loadedFrames = loadFrames(o);
                processFrames(o, loadedFrames);
                clear loadedFrames;
                display('');
                display('');
            end
        end %Tracking() Constructor

        function processFrames(o, loadedFrames)
            if loadedFrames.nFrames > 1
                j= 1;
                display(sprintf('%d Frames loaded : mmReadLoadTime=%f, timecode=%f,
recordingtime=%f', loadedFrames.nFrames,
loadedFrames.loadTime,loadedFrames.video.times(1,1), o.currentFrameStartTime));
                for k=1 : loadedFrames.nFrames-1
                    a=loadedFrames.video.frames(1,k).cdata-
loadedFrames.video.frames(1,k+1).cdata;
                    BW=im2bw(a,0.05);
                    [B,L] = bwboundaries(BW,'noholes');
                    STATS = regionprops(BW, 'centroid');
                    s=size(STATS);
                    x1=0;
                    y1=0;
                    count = s(1,1);
                    for i=1:count
                        xy=STATS(i,1).Centroid;
                        x1=xy(1,1)+x1;
                        y1=xy(1,2)+y1;
                    end
                    if(x1>0)
                        x1=x1/count;
                    end
                    if(y1>0)
                        y1=y1/count;
                    end
                    % BW=medfilt2(BW);
```

```
          BW=medfilt2(BW);
          v=var(BW);
          vv=BW';
          x_variance=v;
          v=var(vv);
          %keyboard;
          y_variance=v;
          s_x=find(x_variance>0.008);
          s_y=find(y_variance>0.008);
          size_s_x=size(s_x);
          size_s_y=size(s_y);
          if(size_s_x(1,2)>10 && size_s_y(1,2)>10)
            x=floor(mean(s_x));
            y=floor(mean(s_y));
            if(size(find(x_variance),2)>20 && size(find(y_variance),2)>20)
              coords(1, j) = x;
              coords(2, j) = y;
              if j == 1
                coords(3,j) = 0;
              else
                coords(3, j) = loadedFrames.video.times(1,k) -
loadedFrames.video.times(1, j-1);
              end
              coords(4, j) = k;
              j = j+1;
            end
          end
        end
        if j > 1
          m = j -1;
          x = 0;
          y = 0;
          vx = 0;
          vy = 0;
          for i = 1 : m
            if i > 1
              vx = vx + (coords(1, i) - coords(1, i-1))/coords(3, i);
              vy = vy + (coords(2, i) - coords(2, i-1))/coords(3, i);
            end
            x = coords(1, i) + x;
            y = coords(2, i) + y;
          end
          x = x/m;
          y = y/m;
```

```
                vx = vx/m;
                vy = vy/m;
                x3 = x - vx*0.1;
                y3 = y - vy*0.1;
                %x = coords(1, m) + coords(5, m)*0.2;
                %y = coords(2, m) + coords(6, m)*0.2;
                x2 = x3 - loadedFrames.video.width/2;
                y2 = y3 - loadedFrames.video.height/2;
                o.cameraController.moveCamera(x2, y2);
                display(sprintf('x = %d, y = %d',x3, y3));
                display(sprintf('Average x = %d, y = %d',x, y));
                display(sprintf('Average vx = %d, vy = %d',vx, vy));
            end
        end
    end

    function video = loadFrames(o)
        frameEndTime = o.currentFrameStartTime + o.videoAccessDuration;
        a = tic;
        n = 0;
        display(sprintf('Start Access Time Frame (%f - %f)', o.currentFrameStartTime,
frameEndTime));
        while true
            fileNameChanged = checkFileName(o);
            if fileNameChanged
                frameEndTime = o.currentFrameStartTime + o.videoAccessDuration;
                display(sprintf('Start Access Time Frame (%f - %f)', o.currentFrameStartTime,
frameEndTime));
            end
            video = VideoFrames(o.currentFileName, o.currentFrameStartTime,
frameEndTime);
            if video.nFrames == 0
                %nFrames is zero try to access frames again for the
                %same time
                n = n+1;
            else
                break;
            end
        end
        time = toc(a);
        display(sprintf('Finish Access Time Frame (%f - %f), FailedAttempts=%d
TotalAccessTime=%f nFrames=%d', o.currentFrameStartTime, frameEndTime, n, time,
video.nFrames));
    end
```

```
        function fileNameChanged = checkFileName(o)
            %check to see if a new recording has started
            fileNameChanged = false;
            fileName =  getNewestFileName(o);
            if ~strcmp(fileName, o.currentFileName)
                %there is a new file recording, reset the filename and
                %start the recording time again
                o.newRecordingStarted = true;
                o.currentFileName = fileName;
                o.currentFileRecordStartTime = tic;
                o.currentFrameStartTime = 0;
                display(strcat('New recording found at: ', fileName));
                fileNameChanged = true;
            end
        end

        function filename = getNewestFileName(o)
            n = 0;
            d = 0;
            while n == 0
                D = dir(o.directory);
                for i = 1: size(D, 1)
                    name = D(i,1).name;
                    t = findstr('asf', name);
                    if size(t) > 0
                        temp_d = str2double(name(2:9));
                        if temp_d >= d
                            if temp_d > d
                                n = 0;
                                d = temp_d;
                            end
                            temp_n = str2double(name(11:16));
                            if temp_n > n
                                n = temp_n;
                            end
                        end
                    end
                end %getNewestFileName
                filename = strcat(o.directory, name);
            end
        end
    end %methods
end %Tracking
```

Video Frame Grab

```
classdef VideoFrames < handle
    properties(SetAccess = private, GetAccess = public)
        nFrames;
        loadTime;
        video;
    end %properties
    methods
        function o = VideoFrames(filename, startTime, endTime)
            t = tic;
            o.video = mmread(filename, [], [startTime, endTime], false, true);
            o.loadTime = toc(t);
            try
                o.nFrames = size(o.video.frames, 2);
            catch
                o.nFrames = 0;
            end
        end %constructor
        function delete(o)
           clear o.video;
        end
    end %methods
end %VideoFrames
```

Arduino Server Code:

```
/* Analog and Digital Input and Output Server for MATLAB    */
/* Giampiero Campa, Copyright 2009 The MathWorks, Inc       */

/* This file is meant to be used with the MATLAB arduino IO
   package, however, it can be used from the IDE environment
   (or any other serial terminal) by typing commands like:

   0e0   : assigns digital pin #4 (e) as input
   0f1   : assigns digital pin #5 (f) as output
   0n1   : assigns digital pin #13 (n) as output

   1c    : reads digital pin #2 (c)
   1e    : reads digital pin #4 (e)
   2n0   : sets digital pin #13 (n) low
   2n1   : sets digital pin #13 (n) high
   2f1   : sets digital pin #5 (f) high
```

```
  2f0   : sets digital pin #5 (f) low
  4j2   : sets digital pin #9 (j) to  50=ascii(2) over 255
  4jz   : sets digital pin #9 (j) to 122=ascii(z) over 255
  3a    : reads analog pin #0 (a)
  3f    : reads analog pin #5 (f)

  5a    : reads status (attached/detached) of servo #1
  5b    : reads status (attached/detached) of servo #2
  6a1   : attaches servo #1
  8az   : moves servo #1 of 122 degrees (122=ascii(z))
  7a    : reads servo #1 angle
  6a0   : detaches servo #1

  A1z   : sets speed of motor #1 to 122 over 255  (122=ascii(z))
  A4A   : sets speed of motor #4 to 65 over 255 (65=ascii(A))
  B1f   : runs motor #1 forward (f=forward)
  B4b   : runs motor #1 backward (b=backward)
  B1r   : releases motor #1 (r=release)

  C12   : sets speed of stepper motor #1 to 50 rpm  (50=ascii(2))
  C2Z   : sets speed of stepper motor #2 to 90 rpm  (90=ascii(Z))
  D1fsz : does 122 steps on motor #1 forward in single (s) mode
  D1biA : does 65 steps on motor #1 backward in interleave (i) mode
  D2fdz : does 122 steps on motor #1 forward in double (d) mode
  D2bmA : does 65 steps on motor #2 backward in microstep (m) mode
  D1r   : releases motor #1 (r=release)
  D2r   : releases motor #2 (r=release)

  R0    : sets analog reference to DEFAULT
  R1    : sets analog reference to INTERNAL
  R2    : sets analog reference to EXTERNAL

  99    : returns script type (1 basic, 2 motor, 3 general) */

#include <AFMotor.h>
#include <Servo.h>

/* define internal for the MEGA as 1.1V (as as for the 328)  */
#if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
#define INTERNAL INTERNAL1V1
#endif

/* create and initialize servos                      */
Servo servo1;
Servo servo2;

/* create and initialize motors                    */
AF_Stepper stm1(200, 1);
AF_Stepper stm2(200, 2);
AF_DCMotor dcm1(1, MOTOR12_64KHZ); // create motor #1, 64KHz pwm
AF_DCMotor dcm2(2, MOTOR12_64KHZ); // create motor #2, 64KHz pwm
AF_DCMotor dcm3(3, MOTOR12_64KHZ); // create motor #3, 64KHz pwm
AF_DCMotor dcm4(4, MOTOR12_64KHZ); // create motor #4, 64KHz pwm

void setup() {
  /* initialize serial                        */
  Serial.begin(115200);
}


void loop() {

  /* variables declaration and initialization          */

  static int  s   = -1;    /* state                    */
  static int  pin = 13;    /* generic pin number         */
  static int  srv =  2;    /* generic servo number        */
  static int  dcm =  4;    /* generic dc motor number      */

  static int  stm =  2;    /* generic stepper motor number   */
  static int  dir =  0;    /* direction (stepper)          */
  static int  sty =  0;    /* style (stepper)             */

  int  val =  0;          /* generic value read from serial */
  int  agv =  0;          /* generic analog value          */
  int  dgv =  0;          /* generic digital value         */

  /* The following instruction constantly checks if anything
     is available on the serial port. Nothing gets executed in
     the loop if nothing is available to be read, but as soon
     as anything becomes available, then the part coded after
     the if statement (that is the real stuff) gets executed */

  if (Serial.available() >0) {

    /* whatever is available from the serial is read here    */
    val = Serial.read();
```

```
/* This part basically implements a state machine that
   reads the serial port and makes just one transition
   to a new state, depending on both the previous state
   and the command that is read from the serial port.
   Some commands need additional inputs from the serial
   port, so they need 2 or 3 state transitions (each one
   happening as soon as anything new is available from
   the serial port) to be fully executed. After a command
   is fully executed the state returns to its initial
   value s=-1                                    */

switch (s) {


  /* s=-1 means NOTHING RECEIVED YET ******************* */
  case -1:

  /* calculate next state when s=-1                   */
  if (val>47 && val<90) {
        /* the first received value indicates the mode
     49 is ascii for 1, ... 90 is ascii for Z
     s=0 is change-pin mode
     s=10 is DI;  s=20 is DO;  s=30 is AI;  s=40 is AO;
     s=50 is servo status; s=60 is aervo attach/detach;
     s=70 is servo read;   s=80 is servo write
     s=90 is query script type (1 basic, 2 motor)
     s=170 is dc motor set speed
     s=180 is dc motor run/release
     s=190 is stepper motor set speed
     s=200 is stepper motor run/release
     s=340 is change analog reference
                                             */
   s=10*(val-48);
  }

  /* the following statements are needed to handle
     unexpected first values coming from the serial (if
     the value is unrecognized then it defaults to s=-1) */
  if ((s>90 && s<170) || (s>200 && s!=340)) {
   s=-1;
  }

  /* the break statements gets out of the switch-case, so
  /* we go back to line 97 and wait for new serial data  */
```

```
break; /* s=-1 (initial state) taken care of          */


/* s=0 or 1 means CHANGE PIN MODE                    */

case 0:
/* the second received value indicates the pin
   from abs('c')=99, pin 2, to abs('t')=116, pin 19    */
if (val>98 && val<117) {
  pin=val-97;              /* calculate pin         */
  s=1; /* next we will need to get 0 or 1 from serial  */
}
else {
  s=-1; /* if value is not a pin then return to -1     */
}
break; /* s=0 taken care of                         */


case 1:
/* the third received value indicates the value 0 or 1 */
if (val>47 && val<50) {
  /* set pin mode                                */
  if (val==48) {
    pinMode(pin,INPUT);
  }
  else {
    pinMode(pin,OUTPUT);
  }
}
s=-1;  /* we are done with CHANGE PIN so go to -1      */
break; /* s=1 taken care of                         */


/* s=10 means DIGITAL INPUT ************************** */

case 10:
/* the second received value indicates the pin
   from abs('c')=99, pin 2, to abs('t')=116, pin 19    */
if (val>98 && val<117) {
  pin=val-97;              /* calculate pin         */
  dgv=digitalRead(pin);     /* perform Digital Input  */
  Serial.println(dgv);      /* send value via serial  */
}
s=-1;  /* we are done with DI so next state is -1      */
```

```
break; /* s=10 taken care of                          */


/* s=20 or 21 means DIGITAL OUTPUT ****************** */

case 20:
/* the second received value indicates the pin
   from abs('c')=99, pin 2, to abs('t')=116, pin 19    */
if (val>98 && val<117) {
  pin=val-97;               /* calculate pin         */
  s=21; /* next we will need to get 0 or 1 from serial */
}
else {
  s=-1; /* if value is not a pin then return to -1     */
}
break; /* s=20 taken care of                          */

case 21:
/* the third received value indicates the value 0 or 1 */
if (val>47 && val<50) {
  dgv=val-48;               /* calculate value       */
      digitalWrite(pin,dgv);     /* perform Digital Output */
}
s=-1;  /* we are done with DO so next state is -1      */
break; /* s=21 taken care of                          */


/* s=30 means ANALOG INPUT *************************** */

case 30:
/* the second received value indicates the pin
   from abs('a')=97, pin 0, to abs('f')=102, pin 6,
   note that these are the digital pins from 14 to 19
   located in the lower right part of the board       */
if (val>96 && val<103) {
  pin=val-97;               /* calculate pin         */
  agv=analogRead(pin);      /* perform Analog Input   */
      Serial.println(agv);      /* send value via serial  */
}
s=-1;  /* we are done with AI so next state is -1      */
break; /* s=30 taken care of                          */


/* s=40 or 41 means ANALOG OUTPUT ******************* */
```

```
case 40:
/* the second received value indicates the pin
   from abs('c')=99, pin 2, to abs('t')=116, pin 19    */
if (val>98 && val<117) {
  pin=val-97;              /* calculate pin         */
  s=41; /* next we will need to get value from serial  */
}
else {
  s=-1; /* if value is not a pin then return to -1     */
}
break; /* s=40 taken care of                         */


case 41:
/* the third received value indicates the analog value */
analogWrite(pin,val);       /* perform Analog Output  */
s=-1;  /* we are done with AO so next state is -1      */
break; /* s=41 taken care of                         */


/* s=50 means SERVO STATUS (ATTACHED/DETACHED) ******* */

case 50:
/* the second received value indicates the servo number
   from abs('a')=97, servo1, on top, uses digital pin 10
   to abs('b')=98, servo2, bottom, uses digital pin 9  */
if (val>96 && val<99) {
  srv=val-96;              /* calculate srv         */
  if (srv==1) dgv=servo1.attached();    /* read status */
  if (srv==2) dgv=servo2.attached();
  Serial.println(dgv);      /* send value via serial  */
}
s=-1;  /* we are done with servo status so return to -1*/
break; /* s=50 taken care of                         */


/* s=60 or 61 means SERVO ATTACH/DETACH ************** */

case 60:
/* the second received value indicates the servo number
   from abs('a')=97, servo1, on top, uses digital pin 10
   to abs('b')=98, servo2, bottom, uses digital pin 9  */
if (val>96 && val<99) {
 srv=val-96;              /* calculate srv         */
 s=61; /* next we will need to get 0 or 1 from serial */
}
else {
 s=-1; /* if value is not a servo then return to -1   */
}
break; /* s=60 taken care of                        */


case 61:
/* the third received value indicates the value 0 or 1
   0 for detach and 1 for attach                    */
if (val>47 && val<50) {
 dgv=val-48;              /* calculate value       */
 if (srv==1) {
   if (dgv) servo1.attach(10);     /* attach servo 1 */
   else servo1.detach();          /* detach servo 1 */
 }
 if (srv==2) {
   if (dgv) servo2.attach(9);      /* attach servo 2 */
   else servo2.detach();          /* detach servo 2 */
 }
}
s=-1;  /* we are done with servo attach/detach so -1   */
break; /* s=61 taken care of                        */


/* s=70 means SERVO READ ***************************** */

case 70:
/* the second received value indicates the servo number
   from abs('a')=97, servo1, on top, uses digital pin 10
   to abs('b')=98, servo2, bottom, uses digital pin 9  */
if (val>96 && val<99) {
 srv=val-96;              /* calculate servo number */
 if (srv==1) agv=servo1.read();        /* read value */
 if (srv==2) agv=servo2.read();
     Serial.println(agv);      /* send value via serial  */
}
s=-1;  /* we are done with servo read so go to -1 next */
break; /* s=70 taken care of                        */


/* s=80 or 81 means SERVO WRITE   ******************** */
```

```
case 80:
/* the second received value indicates the servo number
   from abs('a')=97, servo1, on top, uses digital pin 10
   to abs('b')=98, servo2, bottom, uses digital pin 9  */
if (val>96 && val<99) {
  srv=val-96;              /* calculate servo number */
  s=81; /* next we will need to get value from serial  */
}
else {
  s=-1; /* if value is not a servo then return to -1   */
}
break; /* s=80 taken care of                         */


case 81:
/* the third received value indicates the servo angle  */
if (srv==1) servo1.write(val);         /* write value */
if (srv==2) servo2.write(val);
s=-1;  /* we are done with servo write so go to -1 next*/
break; /* s=81 taken care of                         */



/* s=90 means Query Script Type (1 basic, 2 motor)     */
case 90:
if (val==57) {
  /* if string sent is 99  send script type via serial */
  Serial.println(2);
}
s=-1;  /* we are done with this so next state is -1    */
break; /* s=90 taken care of                         */



/* s=170 or 171 means DC MOTOR SET SPEED  ************ */

case 170:
/* the second received value indicates the motor number
   from abs('1')=49, motor1, to abs('4')=52, motor4    */
if (val>48 && val<53) {
  dcm=val-48;              /* calculate motor number */
  s=171; /* next we will need to get value from serial */
}
```

```
else {
  s=-1; /* if value is not a motor then return to -1   */
}
break; /* s=170 taken care of                         */


case 171:
/* the third received value indicates the motor speed  */
if (dcm==1) dcm1.setSpeed(val);
if (dcm==2) dcm2.setSpeed(val);
if (dcm==3) dcm3.setSpeed(val);
if (dcm==4) dcm4.setSpeed(val);
s=-1;  /* we are done with servo write so go to -1 next*/
break; /* s=171 taken care of                         */



/* s=180 or 181 means DC MOTOR RUN/RELEASE  ********** */
case 180:
/* the second received value indicates the motor number
   from abs('1')=49, motor1, to abs('4')=52, motor4    */
if (val>48 && val<53) {
  dcm=val-48;              /* calculate motor number */
  s=181; /* next we will need to get value from serial */
}
else {
  s=-1; /* if value is not a motor then return to -1   */
}
break; /* s=180 taken care of                         */

case 181:
/* the third received value indicates forward, backward,
   release, with characters 'f', 'b', 'r', respectively,
   that have ascii codes 102, 98 and 114              */
if (dcm==1) {
  if (val==102) dcm1.run(FORWARD);
  if (val==98)  dcm1.run(BACKWARD);
  if (val==114) dcm1.run(RELEASE);
}
if (dcm==2) {
  if (val==102) dcm2.run(FORWARD);
  if (val==98)  dcm2.run(BACKWARD);
  if (val==114) dcm2.run(RELEASE);
}
```

```
if (dcm==3) {
  if (val==102) dcm3.run(FORWARD);
  if (val==98)  dcm3.run(BACKWARD);
  if (val==114) dcm3.run(RELEASE);
}
if (dcm==4) {
  if (val==102) dcm4.run(FORWARD);
  if (val==98)  dcm4.run(BACKWARD);
  if (val==114) dcm4.run(RELEASE);
}
s=-1;  /* we are done with motor run so go to -1 next  */
break; /* s=181 taken care of                        */



/* s=190 or 191 means STEPPER MOTOR SET SPEED  ******* */

case 190:
/* the second received value indicates the motor number
   from abs('1')=49, motor1, to abs('2')=50, motor4    */
if (val>48 && val<51) {
  stm=val-48;              /* calculate motor number */
  s=191; /* next we will need to get value from serial */
}
else {
  s=-1; /* if value is not a stepper then return to -1 */
}
break; /* s=190 taken care of                        */


case 191:
/* the third received value indicates the speed in rpm */
if (stm==1) stm1.setSpeed(val);
if (stm==2) stm2.setSpeed(val);

s=-1;  /* we are done with set speed so go to -1 next  */
break; /* s=191 taken care of                        */



/* s=200 or 201 means STEPPER MOTOR STEP/RELEASE  **** */

case 200:
/* the second received value indicates the motor number
```

```
   from abs('1')=49, motor1, to abs('2')=50, motor4    */
if (val>48 && val<51) {
  stm=val-48;              /* calculate motor number */
  s=201;           /* we still need stuff from serial */
}
else {
  s=-1; /* if value is not a motor then return to -1   */
}
break; /* s=200 taken care of                          */


case 201:
/* the third received value indicates forward, backward,
   release, with characters 'f', 'b', 'r', respectively,
   that have ascii codes 102, 98 and 114              */
switch (val) {

  case 102:
  //dir=FORWARD;
  s=202;
  break;

  case 98:
  //dir=BACKWARD;
  s=202;
  break;

  case 114: /* release and return to -1 here          */
  if (stm==1) stm1.release();
  if (stm==2) stm2.release();
  s=-1;
  break;

  default:
  s=-1;  /* unrecognized  character, go to -1         */
  break;
}
break; /* s=201 taken care of                          */


case 202:
/* the third received value indicates the style, single,
   double, interleave, microstep, 's', 'd', 'i', 'm'
   that have ascii codes 115,100,105 and 109          */
```

```
switch (val) {

   case 115:
//       sty=SINGLE;
   s=203;
   break;

   case 100:
  // sty=DOUBLE;
   s=203;
   break;

   case 105:
   //sty=INTERLEAVE;
   s=203;
   break;

   case 109:
  // sty=MICROSTEP;
   s=203;
   break;

   default:
   s=-1;  /* unrecognized  character, go to -1          */
   break;
 }
 break; /* s=201 taken care of                         */


case 203:
/* the last received value indicates the number of
   steps,                                        */
//if (stm==1) stm1.step(val,dir,sty);  /* do the steps   */
//if (stm==2) stm2.step(val,dir,sty);
s=-1;       /* we are done with step so go to -1 next */
break;      /* s=203 taken care of                   */



/* s=340 or 341 means ANALOG REFERENCE *************** */

case 340:
/* the second received value indicates the reference,
   which is encoded as is 0,1,2 for DEFAULT, INTERNAL
```

```
       and EXTERNAL, respectively                    */

     switch (val) {

       case 48:
       analogReference(DEFAULT);
       break;

       case 49:
       analogReference(INTERNAL);
       break;

       case 50:
       analogReference(EXTERNAL);
       break;

       default:             /* unrecognized, no action  */
       break;
     }
     s=-1;  /* we are done with this so next state is -1    */
     break; /* s=341 taken care of                     */
     /* ******* UNRECOGNIZED STATE, go back to s=-1 ******* */
     default:
     /* we should never get here but if we do it means we
        are in an unexpected state so whatever is the second
        received value we get out of here and back to s=-1  */
     s=-1;  /* go back to the initial state, break unneeded */
    } /* end switch on state s                         */

   } /* end if serial available                        */

  } /* end loop statement                               */
```