A project report

on

SURVEILLANCE BASED ON TRACKING & TARGETING

SUBMITTED BY

BAHUDDIN HAMDULAY

NITISH GUPTA

PRABIN PREMRAJAN

SIDDHARTH SHARMA

UNDER THE GUIDANCE OF

PROF. AJIT SARAF

DEPARTMENT OF ELECTRONICS ENGINEERING

PILLAI'S INSTITUTE OF INFORMATION TECHNOLOGY, ENGINEERING, MEDIA STUDIES & RESEARCH

NEW PANVEL, INDIA – 410 206

UNIVERSITY OF MUMBAI

Academic Year 2013 – 14

Chapter 1
INTRODUCTION

1.1. Introduction to Video Processing:

Object tracking is an important task within the field of computer vision. The proliferation of high powered computers, the availability of high quality and inexpensive video cameras, and the increasing need for automated video analysis has generated a great deal of interest in object tracking algorithms. There are three key steps in video analysis: detection of interesting moving objects, tracking of such objects from frame to frame, and analysis of object tracks to recognize their behavior. Therefore, the use of object tracking is pertinent in the tasks of:

1. Motion-based recognition, that is, human identification based on gait, automatic object detection, etc.;
2. Automated surveillance, that is, monitoring a scene to detect suspicious activities or unlikely events;
3. Video indexing, that is, automatic annotation and retrieval of the videos in multimedia databases;
4. human-computer interaction, that is, gesture recognition, eye gaze tracking for data input to computers, etc.;
5. Traffic monitoring, that is, real-time gathering of traffic statistics to direct traffic flow.
6. Vehicle navigation that is, video-based path planning and obstacle avoidance capabilities.

In its simplest form, tracking can be defined as the problem of estimating the trajectory of an object in the image plane as it moves around a scene. In other words, a tracker assigns consistent labels to the tracked objects in different frames of a video. Additionally, depending on the tracking domain, a tracker can also provide object-centric information, such as orientation, area, or shape of an object.

1.2. History of Video Processing:

Video surveillance systems have long been in use to monitor security sensitive areas. The making of video surveillance systems “smart” requires fast, reliable and robust algorithms for moving object detection, classification, tracking and activity analysis.

Moving object detection is the basic step for further analysis of video. It handles segmentation of moving objects from stationary background objects. Commonly used techniques for object detection are background subtraction, statistical models, temporal differencing and optical flow. Currently, there are two major approaches towards moving object classification, which are shape-based and motion-based methods. The next step in the video analysis is tracking, which can be simply defined as the creation of temporal correspondence among detected objects from frame to frame. This procedure provides temporal identification of the segmented regions and generates cohesive information about the objects in the monitored area such as trajectory, speed and direction. The output produced by tracking step is generally used to support and enhance motion segmentation, object classification and higher level activity analysis. The outputs of these algorithms can be used both for providing the human operator with high level data to help him to make the decisions more accurately and in a shorter time

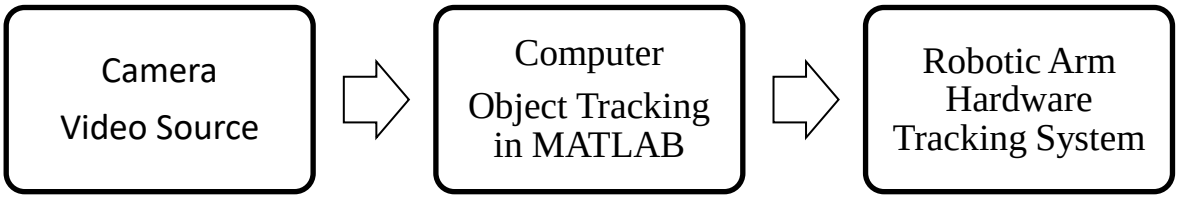


Fig 1.Block Diagram

1.3. System Block Diagram

This is a simple yet efficient block diagram representation of the proposed video processing system. The camera is the video source which is used to capture the video to be supervised. This video as a continuous frame of images is passed to the computer. The person in the surveillance room will be observing this video. Upon finding a suspicious object, he will start the tracking system. This system will ask him to select the object to be tracked. Then the system will track the object continuously. This tracking information will also be passed on to the hardware unit i.e. the ROBOTIC ARM which will track the object on the scene.

Chapter 2
LITERATURE SURVEY

There have been a number of surveys about object detection, classification, tracking and activity analysis in the literature. The survey we present here covers only that work which are in the same context as our study. However, for comprehensive completeness, we also give brief information on some techniques which are used for similar tasks that are not covered in our study. Although, some steps require interchange of information with other levels, this framework provides a good structure for the discussion throughout this survey.

2.1 Moving Object Detection

Each application that benefit from smart video processing has different needs, thus requires different treatment. However, they have something in common moving objects. Thus, detecting regions that correspond to moving objects such as people and vehicles in video is the first basic step of almost every vision system since it provides a focus of attention and simplifies the processing on subsequent analysis steps. Due to dynamic changes in natural scenes such as sudden illumination and weather changes, repetitive motions that cause clutter (tree leaves moving in blowing wind), motion detection is a difficult problem to process reliably. Frequently used techniques for moving object detection are background subtraction, statistical methods, temporal differencing and optical flow whose descriptions are given below.

2.1.1. Background Subtraction

Background subtraction is particularly a commonly used technique for motion segmentation in static scenes. It attempts to detect moving regions by subtracting the current image pixel-by-pixel from a reference background image that is created by averaging images over time in an initialization period. The pixels where the difference is above a threshold are classified as foreground. After creating a foreground pixel map, some morphological post processing operations such as erosion, dilation and closing are performed to reduce the effects of noise and enhance the detected regions. The reference background is updated with new images over time to adapt to dynamic scene changes. There are different approaches to this basic scheme of background subtraction in terms of foreground region detection, background maintenance and post processing.

In Heikkila and Silven uses the simple version of this scheme where a pixel at location (x, y) in the current image I_t is marked as foreground if

$$|I_t(x, y) - B_t(x, y)| > \tau \qquad (2.1)$$

is satisfied where τ is a predefined threshold. The background image BT is updated by the use of an Infinite Impulse Response (IIR) filter as follows:

$$B_{t+1} = \alpha I_t + (1 - \alpha) B_t \quad (2.2)$$

The foreground pixel map creation is followed by morphological closing and the elimination of small-sized regions. Although background subtraction techniques perform well at extracting most of the relevant pixels of moving regions even they stop, they are usually sensitive to dynamic changes when, for instance, stationary objects uncover the background (e.g. a parked car moves out of the parking lot) or sudden illumination changes occur.

2.1.2. Statistical Methods

More advanced methods that make use of the statistical characteristics of individual pixels have been developed to overcome the shortcomings of basic background subtraction methods. These statistical methods are mainly inspired by the background subtraction methods in terms of keeping and dynamically updating statistics of the pixels that belong to the background image process. Foreground pixels are identified by comparing each pixel's statistics with that of the background model. This approach is becoming more popular due to its reliability in scenes that contain noise, illumination changes and shadow.

The W4 system uses a statistical background model where each pixel is represented with its minimum (M) and maximum (N) intensity values and maximum intensity difference (D) between any consecutive frames observed during initial training period where the scene contains no moving objects. A pixel in the current image I_t is classified as foreground if it satisfies

$$|M(x, y) - It(x, y)| > D(x, y) \text{ or } |N(x, y) - It(x, y)| > D(x, y) \quad (2.3)$$

After thresholding, a single iteration of morphological erosion is applied to the detected foreground pixels to remove one-pixel thick noise. In order to grow the eroded regions to their original sizes, a sequence of erosion and dilation is performed on the foreground pixel map. Also, small-sized regions are eliminated after applying connected component labeling to find the regions. The statistics of the background pixels that belong to the non-moving regions of current image are updated with new image data.

As another example of statistical methods, Stauffer and Grimson described an adaptive background mixture model for real-time tracking. In their work, every pixel is

separately modeled by a mixture of Gaussians which are updated online by incoming image data. In order to detect whether a pixel belongs to a foreground or background process, the Gaussian distributions of the mixture model for that pixel are evaluated.

2.2. Object Classification

Moving regions detected in video may correspond to different objects in real-world such as pedestrians, vehicles, clutter, etc. It is very important to recognize the type of a detected object in order to track it reliably and analyze its activities correctly. Currently, there are two major approaches towards moving object classification which are shape-based and motion-based methods. Shape-based methods make use of the objects' 2D spatial information whereas motion-based methods use temporally tracked features of objects for the classification solution.

2.2.1. Shape-based Classification

Common features used in shape-based classification schemes are the bounding rectangle, area, silhouette and gradient of detected object regions.

The approach presented in makes use of the objects' silhouette contour length and area information to classify detected objects into three groups: human, vehicle and other. The method depends on the assumption that humans are, in general, smaller than vehicles and have complex shapes. Dispersedness is used as the classification metric and it is defined in terms of object's area and contour length (perimeter) as follows:

$$\text{Dispersedness} = \text{Perimeter}^2/\text{Area} \qquad (2.4)$$

Classification is performed at each frame and tracking results are used to improve temporal classification consistency.

The classification method developed by Collins uses view dependent visual features of detected objects to train a neural network classifier to recognize four classes: human, human group, vehicle and clutter. The inputs to the neural network are the dispersedness, area and aspect ratio of the object region and the camera zoom magnification. Like the previous method, classification is performed at each frame and results are kept in a histogram to improve temporal consistency of classification. Saptharishi propose a classification scheme which uses a logistic linear neural network trained with Differential Learning to recognize two classes: vehicle and people. Papageorgiou et al. presents a method that makes use of the

Support Vector Machine classification trained by wavelet transformed object features (edges) in video images from a sample pedestrian database. This method is used to recognize moving regions that correspond to humans.

Another classification method proposed by Brodsky uses a Radial Basis Function (RBF) classifier which has a similar architecture like a three-layer back-propagation network. The input to the classifier is the normalized gradient image of the detected object regions.

2.2.2. Motion-based Classification

Some of the methods in the literature use only temporal motion features of objects in order to recognize their classes. In general, they are used to distinguish non-rigid objects (e.g. human) from rigid objects (e.g. vehicles). As an object that exhibits periodic motion evolves, its self-similarity measure also shows a periodic motion. The method exploits this clue to categorize moving objects using periodicity. Optical flow analysis is also useful to distinguish rigid and non-rigid objects. A. J. Lipton proposed a method that makes use of the local optical flow analysis of the detected object regions. It is expected that non-rigid objects such as humans will present high average residual flow whereas rigid objects such as vehicles will present little residual flow. Also, the residual flow generated by human motion will have a periodicity. By using this cue, human motion, thus humans, can be distinguished from other objects such as vehicles.

2.3. Object Detection and Tracking

The overview of our real time video object detection, classification and tracking system is shown in Figure 2. The proposed system is able to distinguish transitory and stopped foreground objects from static background objects in dynamic scenes; detect and distinguish left and removed objects; classify detected objects into different groups such as human, human group and vehicle; track objects and generate trajectory information even in multi-occlusion cases and detect fire in video imagery. In this and following chapters we describe the computational models employed in our approach to reach the goals specified above. Our system is assumed to work real time as a part of a video-based surveillance system. The computational complexity and even the constant factors of the algorithms we use are important for real time performance. Hence, our decisions on selecting the computer vision algorithms for various problems are affected by their computational run time performance as well as quality. Furthermore, our systems use is limited only to stationary

cameras and video inputs from Pan/Tilt/Zoom cameras where the view frustum may change arbitrarily are not supported.

The system is initialized by feeding video imagery from a static camera monitoring a site. Most of the methods are able to work on both color and monochrome video imagery. The first step of our approach is distinguishing foreground objects from stationary background. To achieve this, we use a combination of adaptive background subtraction and low-level image post-processing methods to create a foreground pixel map at every frame. This signal is scaled, normalized and compared with pre-labeled signals in a template database to decide on the type of the object. The output of the tracking step is used to attain temporal consistency in the classification step.

The object tracking algorithm utilizes extracted object features together with a correspondence matching scheme to track objects from frame to frame. The color histogram of an object produced in previous step is used to match the correspondences of objects after an occlusion event. The output of the tracking step is object trajectory information which is used to calculate direction and speed of the objects in the scene. After gathering information on objects' features such as type, trajectory, size and speed various high level processing can be applied on these data. A possible use is real-time alarm generation by pre-defining event predicates such as "A human moving in direction d at speed more than s causes alarm a1." or "A vehicle staying at location l more than t seconds causes alarm a2." Another opportunity we may make use of the produced video object data is to create an index on stored video data for offline smart search. Both alarm generation and video indexing are critical requirements of a visual surveillance system to increase response time to forensic events.

2.3.1. Object Detection

Distinguishing foreground objects from the stationary background is both a significant and difficult research problem. Almost in the entire visual surveillance systems first step is detecting foreground objects. This both creates a focus of attention for higher processing levels such as tracking, classification and behavior understanding and reduces computation time considerably since only pixels belonging to foreground objects need to be dealt with. Short and long term dynamic scene changes such as repetitive motions (e. g. waiving tree leaves), light reflectance, shadows, camera noise and sudden illumination variations make reliable and far object detection difficult.

Hence, it is important to pay necessary attention to object detection step to have reliable, robust and fast visual surveillance system. The system diagram of our object detection method is shown in Figure 2.

Our method depends on a six stage process to extract objects with their features in video imagery. The first step is the background scene initialization. There are various techniques used to model the background scene in the literature. In order to evaluate the quality of different background scene models for object detection and to compare run-time performance, we implemented three of these models which are adaptive background subtraction, temporal frame differencing and adaptive online Gaussian mixture model. The background scene related parts of the system is isolated and its coupling with other modules is kept minimum to let the whole detection system to work flexibly with any one of the background models.

Next step in the detection method is detecting the foreground pixels by using the background model and the current image from video. This pixel-level detection process is dependent on the background model in use and it is used to update the background model to adapt to dynamic scene changes. Also, due to camera noise or environmental effects the detected foreground pixel map contains noise. Pixel-level post-processing operations are performed to remove noise in the foreground pixels. Once we get the filtered foreground pixels, in the next step, connected regions are found by using a connected component labeling algorithm and objects' bounding rectangles are calculated. The labeled regions may contain near but disjoint regions due to defects in foreground segmentation process. Hence, it is experimentally found to be effective to merge those overlapping isolated regions. Also, some relatively small regions caused by environmental noise are eliminated in the region-level post-processing step. In the final step of the detection process, a number of object features are extracted from current image by using the foreground pixel map. These features are the area, center of mass and color histogram of the regions corresponding to objects.

2.3.2. Object Tracking

The aim of object tracking is to establish a correspondence between objects or object parts in consecutive frames and to extract temporal information about objects such as trajectory, posture, speed and direction. It is a crucial part of smart surveillance systems since without object tracking, the system could not extract cohesive temporal information about objects and higher level behavior analysis steps would not be possible. On the other hand,

inaccurate foreground object segmentation due to shadows, reflectance and occlusions makes tracking a difficult research problem. We used an object level tracking algorithm in our system. That is, we do not track object parts, such as limbs of a human, but we track objects as a whole from frame to frame. The information extracted by this level of tracking is adequate for most of the smart surveillance applications. The tracking method we have developed is inspired by the study presented in. Furthermore, our tracking algorithm detects object occlusion and distinguishes object identities after the split of occluded objects. By analyzing the object trajectory information, our tracking system is able to detect left and removed objects as well.

2.3.3. Occlusion Handling

Most of the object detection methods are not able to detect occlusions among objects. Therefore, special techniques are required to continue tracking objects even in occlusion cases. Our tracking system uses simple heuristics to detect object occlusions and occlusion group splits and to distinguish object identities (which object is which?) after occlusions. Details of these three steps are described in the following section.

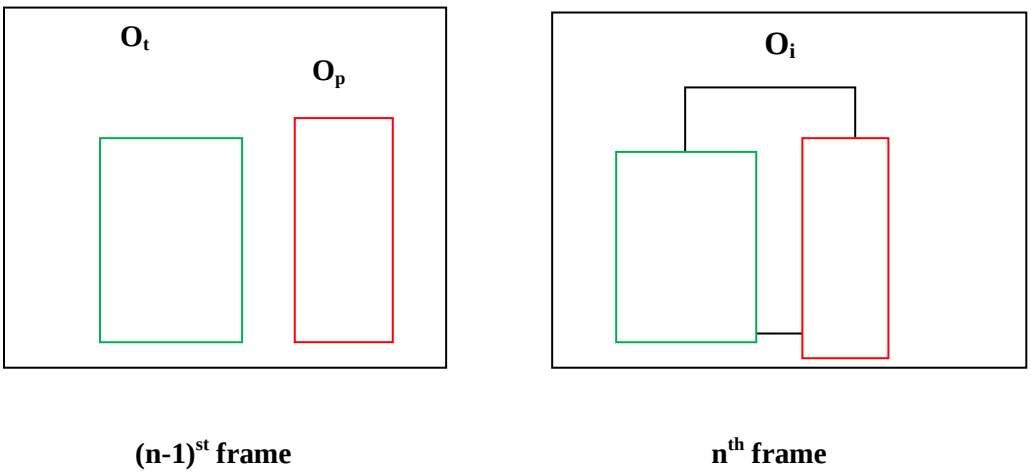


Fig 2.Occlusion Detection sample case

We make use of a simple assumption in detecting occlusions. When an object O_p is found to disappear by the initial object matching algorithm, we check whether there is a new object O_i whose bounding box is overlapping with that of Op and which is matched to a previous object O_t such a case, it is highly possible that O_p and O_t are occluded with each

other and formed a new object O_i such a case, we do not delete object O_p from the previous objects list but mark it as occluded. We create an occlusion group from the objects that are occluded with each other and assign a new occlusion group ID to these objects. For the case if one of the occluding objects has already an occlusion group ID, we merge these different occlusion groups into one. We also, store the pre-occlusion color histograms of objects in order to use in identification process after a split.

2.4. Basic difference between Stepper and Servo motor

Stepper and servo motors differ in two key ways, in their basic construction and how they are controlled. Stepper motors have a large number of poles, magnetic pairs of north and south poles generated either by a permanent magnet or an electric current, typically 50 to 100 poles. In comparison, servo motors have very few poles, often 4 to 12 in total. Each pole offers a natural stopping point for the motor shaft. The greater number of poles allows a stepper motor to move accurately and precisely between each pole and allows a stepper to be operated without any position feedback for many applications. Servo motors often require a position encoder to keep track of the position of the motor shaft, especially if precise movements are required.

Driving a stepper motor to a precise position is much simpler than driving a servo motor. With a stepper motor, a single drive pulse will move the motor shaft one step, from one pole to the next. Since the step size of a given motor is fixed at a certain amount of rotation, moving to a precise position is simply a matter of sending the right number of pulses. In contrast servo motors read the difference between the current encoder position and the position they were commanded to and just the current required moving to the correct position. With today's digital electronics, stepper motors are much easier to control than servo motors.

2.4.1. Stepper motor advantages:

Stepper motors offer several advantages over servo motors beyond the larger number of poles and easier drive control. The design of the stepper motor provides a constant holding torque without the need for the motor to be powered. The torque of a stepper motor at low speeds is greater than a servo motor of the same size. One of the biggest advantages of stepper motors is their relatively low cost and availability.

2.4.2. Servo motor advantages:

For applications where high speed and high torque is needed, servo motors shine. Stepper motors peak around speeds of 2,000 RPM, while servo motors are available many times faster. Servo motors also maintain their torque rating at high speed, up to 90% of the rated torque is available from a servo at high speed. Servo motors are also more efficient than stepper motors with efficiencies between 80-90%. A servo motor can supply roughly twice their rated torque for short periods, providing a well of capacity to draw from when needed. In addition, servo motors are quite, available in AC and DC drive, and do not vibrate or suffer from resonance issues.

2.4.3. Stepper Limitations:

For all of their advantages, stepper motors have a few limitations which can cause significant implementation and operation issues depending on your application. Stepper motors do not have any reserve power. In fact, stepper motors lose a significant amount of their torque as they approach their maximum driver speed. A loss of 80% of the rated torque at 90% of the maximum speed is typical. Stepper motors are also not as good as servo motors in accelerating a load. Attempting to accelerate a load too fast where the stepper cannot generate enough torque to move to the next step before the next drive pulse will result in a skipped step and a loss in position. If positional accuracy is essential, either the load on the motor must never exceed its torque or the stepper must be combined with a position encoder to ensure positional accuracy. Stepper motors also suffer from vibration and resonance problems. At certain speeds, partially depending on the load dynamics, a stepper motor may enter resonance and be unable to drive the load. This results in skipped steps, stalled motors, excessive vibration and noise.

2.4.4. Servo Limitations:

Servo motors are capable of delivering more power than stepper motors, but do require much more complex drive circuitry and positional feedback for accurate positioning. Servo motors are also much more expensive than stepper motors and are often harder to find. Servo motors often require gear boxes, especially for lower speed operation. The requirement for a gearbox and position encoder make servo motor designs more mechanically complex and increase the maintenance requirements for the system. To top it all off, servo motors are more expensive than stepper motors before adding on the cost of a position encoder.

2.4.5. Conclusion:

Selecting the best motor for your application depends on a few key design criteria for your system including cost, positional accuracy requirements, torque requirements, drive power availability, and acceleration requirements. Overall, servo motors are best for high speed, high torque applications while stepper motors are better suited for lower acceleration, high holding torque applications.

2.5. Inside Servo

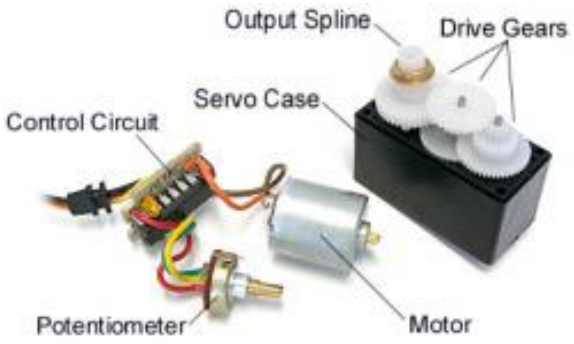


Fig. 3 Inside Servo

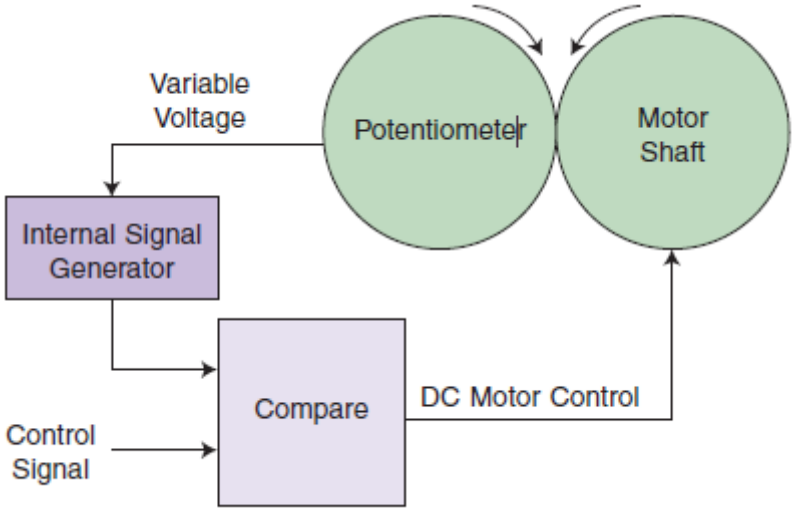


Fig. 4 Block diagram of servo Motor

To fully understand how the servo works, you need to take a look under the hood. Inside there is a pretty simple set-up: a small DC motor, potentiometer, and a control circuit. The motor is attached by gears to the control wheel. As the motor rotates, the potentiometer's resistance changes, so the control circuit can precisely regulate how much movement there is and in which direction. When the shaft of the motor is at the desired position, power supplied

to the motor is stopped. If not, the motor is turned in the appropriate direction. The desired position is sent via electrical pulses through the signal wire. The motor's speed is proportional to the difference between its actual position and desired position. So if the motor is near the desired position, it will turn slowly, otherwise it will turn fast. This is called **proportional control.** This means the motor will only run as hard as necessary to accomplish the task at hand, a very efficient little guy.

2.5.1. Servo and PWM

Pulse-width modulation:

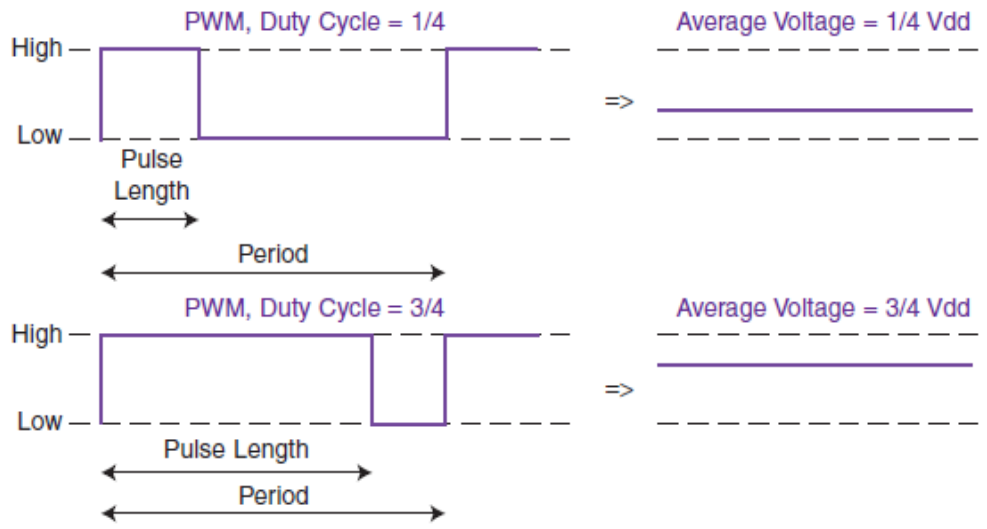


Fig. 5 Pulse width Modulation

Often, when controlling an analog device, the ability to drive a signal with variable power ($P = I \times V$) is needed. For example, you may want to adjust the speed of a dc motor or dim a light-emitting diode (LED). This can be a challenge when the signal is generated by a digital device. Different methods to convert a digital signal to an analog signal exist, one of which is a digital-to-analog converter. Using a converter will add complexity to a project, so generating a variable power signal with existing circuitry is desirable to reduce the number of components.

An easy method to vary the power using a digital signal, when an analog signal isn't available, is by a method called pulse-width modulation (PWM). Instead of controlling the current or voltage of a signal, a pulse-width-modulated signal works by repeatedly pulsing the digital signal high and low at a fast rate. When sufficiently fast, the signal creates an effective average voltage. A shorter PWM period (the length between the rising edges) will

create a cleaner average voltage, because the signal is effectively less “jittery” (i.e., less discharge from the capacitance in the line is needed to smooth the signal), but the minimum period will be limited by the speed of the device generating the signal. The period of the PWM signal is usually constant for a given application, and the high pulse width (the duration of the signal being driven high within one period) is usually variable, so that the average voltage of the signal can be changed. The ratio of high pulse width to period of the signal is called the *duty cycle*. By varying the duty cycle, you can vary the average voltage, as shown in Fig. 2.

The power through a device is proportional to the voltage supplied. Therefore, to decrease the power usage of a device (to dim an LED or to slow a motor), the duty cycle of the PWM signal should be decreased. A PWM signal can be used to limit the power to a device to save energy. This technique is used in many portable devices which have limited battery power. For example, wave a device with an LED back and forth in your hand and you will often see a strobe light effect instead of a straight streak. This blinking is from the pulse-width modulated signal turning the LED on and off. The blinking cannot normally be perceived by the human eye when the LED is stationary, since it is blinking at a rate faster than the eye can perceive. But when moving, the discrete flashes become visible because the LED is only lit up at certain positions as it moves. Some devices, such as servos, do not rely on the power of the signal limited by PWM but instead use the width of the high pulses to transmit information. This is also used by infrared remote controls to transmit data to control a television or radio. Pulse-width-modulated signals may be generated from many digital devices, even ones as simple as an inexpensive timer integrated circuit (such as the 8-pin 555 timer). A versatile yet inexpensive solution for many robotics hobbyists is to use microcontroller for PWM generation. Using a microcontroller has the added advantage of containing all of the control circuitry (needed for analyzing and responding to input) for a simple robot on a single chip.

2.5.2. Controlling a servo:

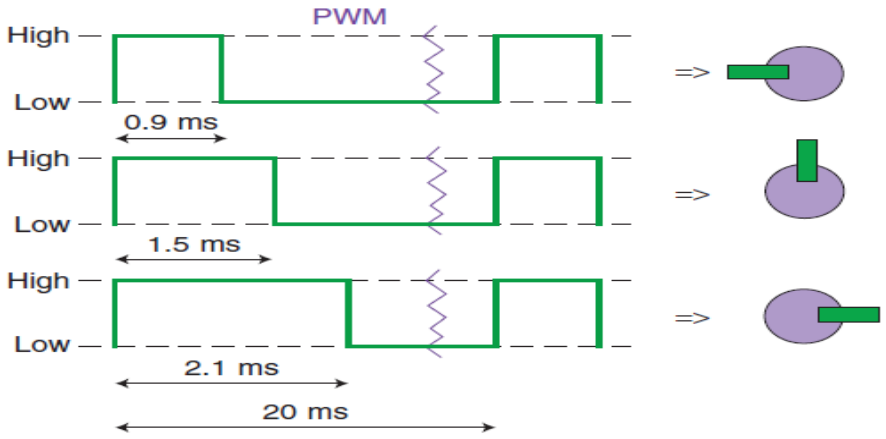


Fig. 6 PWM of Servo Motor

Most servos have three pins: power, ground, and a control signal. The control signal is a pulse-width-modulated input signal whose high pulse width determines the servo's angular position, as shown in Fig. 3. Internally, the servo compares the PWM control signal to an internally generated signal, whose pulse widths are controlled by the potentiometer (which determines the shaft angle) and matches the pulse widths by rotating the motor shaft. For any servo in general, power can be between 4.8 Vdc (volts of direct current) and 6 Vdc. Since the control signal (which draws a maximum of about 20 mA) does not drive the motor directly, an additional benefit of using a servo is that the Arduino (whose output pins can drive up to 25 mA) *can* drive the control signal directly. Most motors draw more than 25 mA of current for operation and, therefore, must be indirectly connected to the Arduino through a current amplifying device, like an Hbridge or a transistor. Typically, servos require a PWM signal with a 20-ms period and a pulse width between 0.9–2.1 ms (0.9 ms corresponds to the minimum angle and 2.1 ms corresponds to the maximum angle); therefore, the middle position is 1.5 ms (the average of the minimum/maximum pulse widths). The general servo has a maximum angle of 180◦. Servos only move a finite angular amount per cycle of the signal, so multiple cycles must be sent before the servo arrives at the correct angle. The speed/power at which the servo moves to a new position is proportional to the distance it needs to travel. So as the servo becomes closer to the target angle, it will gradually slow. The servo will resist change away from the designated angle as long as a signal is applied. Note that the servo's control mechanism will only engage when a signal is applied. If there is no

signal, the servo's motor shaft is not driven by any circuitry and, hence, can be rotated freely, even when power is supplied to the servo.

2.5.3. Conclusions

Microcontrollers offer a simple and inexpensive solution for controlling servo motors for robotics and other electronics projects. Through the use of PWM, the angular position of the servo motor shaft can be conveniently controlled by a microcontroller for a variety of projects. PWM is an easy solution for the control of analog devices in other projects as well. Depending on the features included with the microcontroller you are using, a PWM signal can be generated in a variety of ways.

2.6. About Arduino

2.6.1. What is Arduino?

Arduino is a tool for making computers that can sense and control more of the physical world than your desktop computer. It's an open-source physical computing platform based on a simple microcontroller board, and a development environment for writing software for the board.

Arduino can be used to develop interactive objects, taking inputs from a variety of switches or sensors, and controlling a variety of lights, motors, and other physical outputs. Arduino projects can be stand-alone, or they can communicate with software running on your computer (e.g. Flash, Processing, MaxMSP.) The boards can be assembled by hand or purchased preassembled; the open-source IDE can be downloaded for free.

The Arduino programming language is an implementation of Wiring, a similar physical computing platform, which is based on the Processing multimedia programming environment.

2.6.2. Why Arduino?

There are many other microcontrollers and microcontroller platforms available for physical computing. Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard, and many others offer similar functionality. All of these tools take the messy details of

microcontroller programming and wrap it up in an easy-to-use package. Arduino also simplifies the process of working with microcontrollers, but it offers some advantage for teachers, students, and interested amateurs over other systems:

- Inexpensive - Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than $50.
- Cross-platform - The Arduino software runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.
- Simple, clear programming environment - The Arduino programming environment is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with the look and feel of Arduino
- Open source and extensible software- The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.
- Open source and extensible hardware - The Arduino is based on Atmel's ATMEGA8 and ATMEGA168 microcontrollers. The plans for the modules are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money.

2.6.3. ARDUINO DEVELOPMENT BOARD:

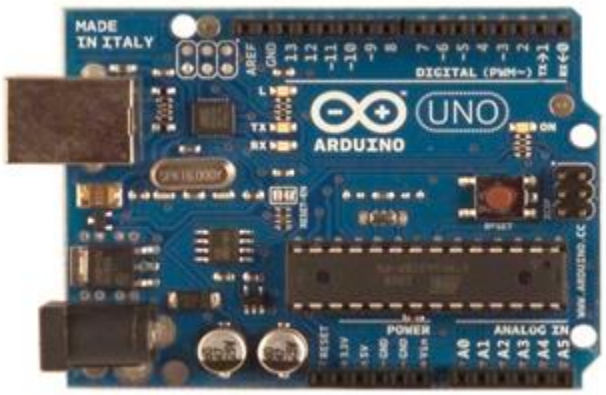


Fig. 7. ARDUINO UNO Board

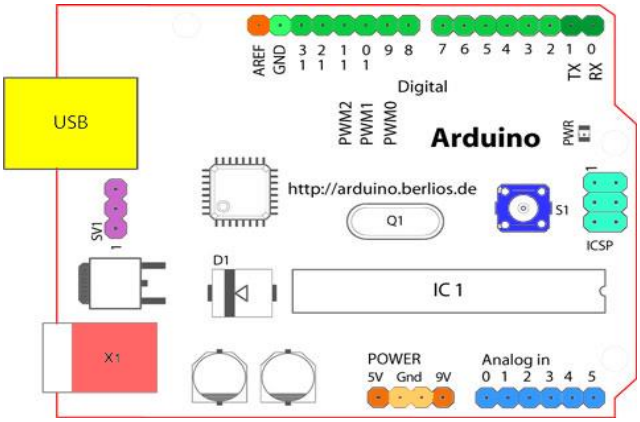


Fig. 8. ARDUINO UNO Schematic.

Starting clockwise from the top center:

- Analog Reference pin (orange)
- Digital Ground (light green)
- Digital Pins 2-13 (green)
- Digital Pins 0-1/Serial In/Out - TX/RX (dark green) - *These pins cannot be used for digital i/o (**digitalRead** and **digitalWrite**) if you are also using serial communication (e.g. **Serial.begin**).*
- Reset Button - S1 (dark blue).
- In-circuit Serial Programmer (blue-green).
- Analog In Pins 0-5 (light blue).
- Power and Ground Pins (power: orange, grounds: light orange) .
- External Power Supply In (9-12VDC) - X1 (pink).

- Toggles External Power and USB Power (place jumper on two pins closest to desired supply) - SV1 (purple).
- USB (used for uploading sketches to the board and for serial communication between the board and the computer; can be used to power the board) (yellow).

2.6.4 ATMEGA328P (used on most recent boards)

FEATURES:

- High performance, low power Atmel® AVR®

8-bit microcontroller

- Advanced RISC architecture

– 131 powerful instructions – most single clock cycle execution

– 32 × 8 general purpose working registers

– Fully static operation

– Up to 20 MIPS throughput at 20MHz

– On-chip 2-cycle multiplier

- High endurance non-volatile memory segments

– 4/8/16 Kbytes of in-system self-programmable flash program memory

– 256/512/512 bytes EEPROM

– 512/1K/1Kbytes internal SRAM

– Write/erase cycles: 10,000 flash/100,000 EEPROM

– Data retention: 20 years at 85°C/100 years at 25°C

– Optional boot code section with independent lock bits In-system programming by on-chip boot program. True read-while-write operation

– Programming locks for software security

- QTouch® library support

– Capacitive touch buttons, sliders and wheels

– QTouch and QMatrix acquisition

–Up to 64 sense channels

- Peripheral features

– Two 8-bit timer/counters with separate prescaler and compare mode

– One 16-bit timer/counter with separate prescaler, compare mode, and capture mode

– Real time counter with separate oscillator

– Six PWM channels

– 8-channel 10-bit ADC in TQFP and QFN/MLF package

– 6-channel 10-bit ADC in PDIP Package

– Programmable serial USART

– Master/slave SPI serial interface

– Byte-oriented 2-wire serial interface (Philips I^2C compatible)

– Programmable watchdog timer with separate on-chip oscillator

– On-chip analog comparator

– Interrupt and wake-up on pin change

- Special microcontroller features

– DebugWIRE on-chip debugs system

– Power-on reset and programmable brown-out detection

– Internal calibrated oscillator

– External and internal interrupt sources

– Five sleep modes: Idle, ADC

noise reduction, power-save, power-down, and standby

- I/O and packages

– 23 programmable I/O lines

– 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF

- Operating voltage:– 1.8V - 5.5V for Atmel ATmega48V/88V/168V– 2.7V - 5.5V for Atmel ATmega48/88/168
- Temperature range:-40°C to 85°C
- Speed grade:

– ATmega48V/88V/168V: 0 - 4MHz @ 1.8V - 5.5V, 0 - 10MHz @ 2.7V - 5.5V

– ATmega48/88/168: 0 - 10MHz @ 2.7V - 5.5V, 0 - 20MHz @ 4.5V - 5.5V

- Low power consumption

– Active mode:

250µA at 1MHz, 1.8V

15µA at 32kHz, 1.8V (including oscillator)

– Power-down mode:

0.1µA at 1.8V

Pin Configuration:

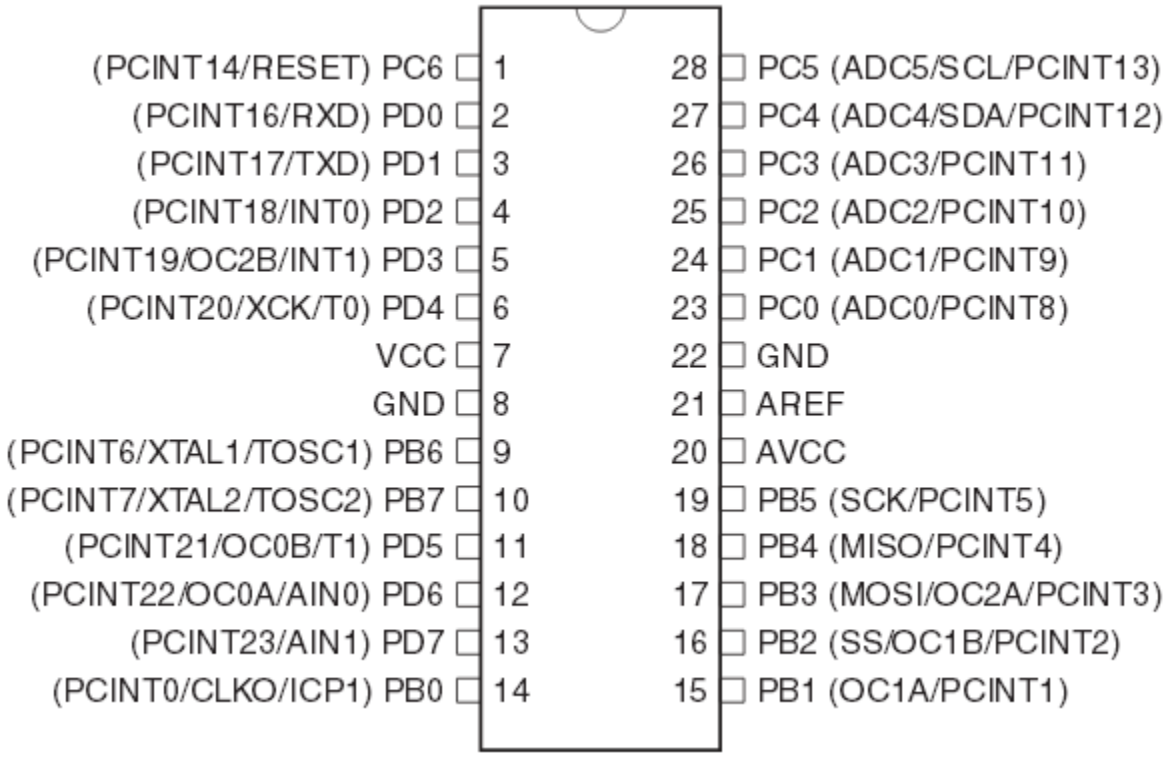


Fig. 9 ATMEGA 328 Pin Configuration

2.6.5. PIN DESCRIPTION

- **Port B (PB7:0) XTAL1/XTAL2/TOSC1/TOSC2**

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running. Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit. Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier. If the Internal Calibrated RC Oscillator is used as chip clock source, PB7..6 is used as TOSC2..1input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

- **Port C (PC5:0)**

Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC5..0 output buffers have symmetrical drive characteristics with both high sink and

source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

- **PC6/RESET**

If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C. If the RSTDISBL Fuse is unprogrammed, PC6 is used as a Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running.

- **Port D (PD7:0)**

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

- **ADC7:6 (TQFP and QFN/MLF package only)**

In the TQFP and QFN/MLF package, ADC7:6 serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

2.7. Using ARDUINO with servo:

2.7.1. Servo library:

This library allows an Arduino board to control RC (hobby) servo motors. Servos have integrated gears and a shaft that can be precisely controlled. Standard servos allow the shaft to be positioned at various angles, usually between 0 and 180 degrees. Continuous rotation servos allow the rotation of the shaft to be set to various speeds.

The Servo library supports up to 12 motors on most Arduino boards and 48 on the Arduino Mega. On boards other than the Mega, use of the library disables analogWrite() (PWM) functionality on pins 9 and 10, whether or not there is a Servo on those pins. On the Mega, up to 12 servos can be used without interfering with PWM functionality; use of 12 to 23 motors will disable PWM on pins 11 and 12.

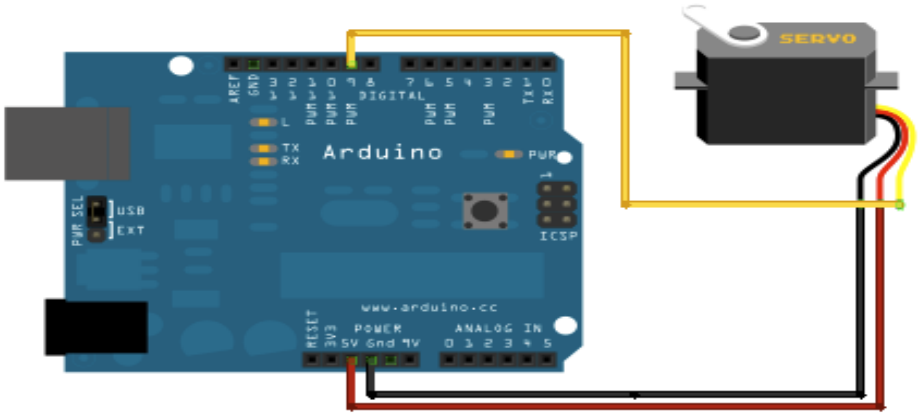


Fig10. Interfacing of servo motor with arduino

2.7.2. Circuit:

Servo motors have three wires: power, ground, and signal. The power wire is typically red, and should be connected to the 5V pin on the Arduino board. The ground wire is typically black or brown and should be connected to a ground pin on the Arduino board. The signal pin is typically yellow, orange or white and should be connected to a digital pin on the Arduino board.

2.8. MATLAB GUI:

A graphical user interface (GUI) is a pictorial interface to a program. A good GUI can make programs easier to use by providing them with a consistent appearance and with intuitive controls like pushbuttons, list boxes, sliders, menus, and so forth. A true GUI includes standard formats for representing text and graphics. The GUI should behave in an understandable and predictable manner, so that a user knows what to expect when he or she performs an action. For example, when a mouse click occurs on a pushbutton, the GUI should initiate the action described on the label of the button.

Operation of GUI:

Each component, and the GUI itself, is associated with one or more userwritten routines known as callbacks. The execution of each callback is triggered by a particular user action such as, mouse click, pushbuttons, toggle buttons, lists, menus, text boxes, selection of a menu item, or the cursor passing over a component and so forth. Clicking the button triggers the execution of a callback. A mouse click or a key press is an event, and the

MATLAB program must respond to each event if the program is to perform its function. For example, if a user clicks on a button, that event must cause the MATLAB code that implements the function of the button to be executed. The code executed in response to an event is known as a callback.

- **Some callback functions:**

Callback Property	Triggering Event
KeyPressFcn	Executes when the user presses a keyboard key and the callback's component or figure has focus.
WindowButtonMotionFcn	Executes when you move the pointer within the figure window.
WindowButtonDownFcn	Executes when you press a mouse button while the pointer is in the figure window.

Chapter 3
PROBLEM STATEMENT
& MOTIVATION

3.1. Problem Statement:

The tracking in broad sense can be an automatic or manual task. However automatic tracking which we proposed earlier was involving complex algorithms and delays in real-time tracking. Also for various security reasons automatic targeting would not be practically realizable. Thus we decided to have a manual tracking system.

3.2. Motivation:

The Horror Night of 26/11 in particular motivated us to work on this Project.

Fig 11. Terrorist attack at CST station

Chapter 4

MATHEMATICAL MODELLING

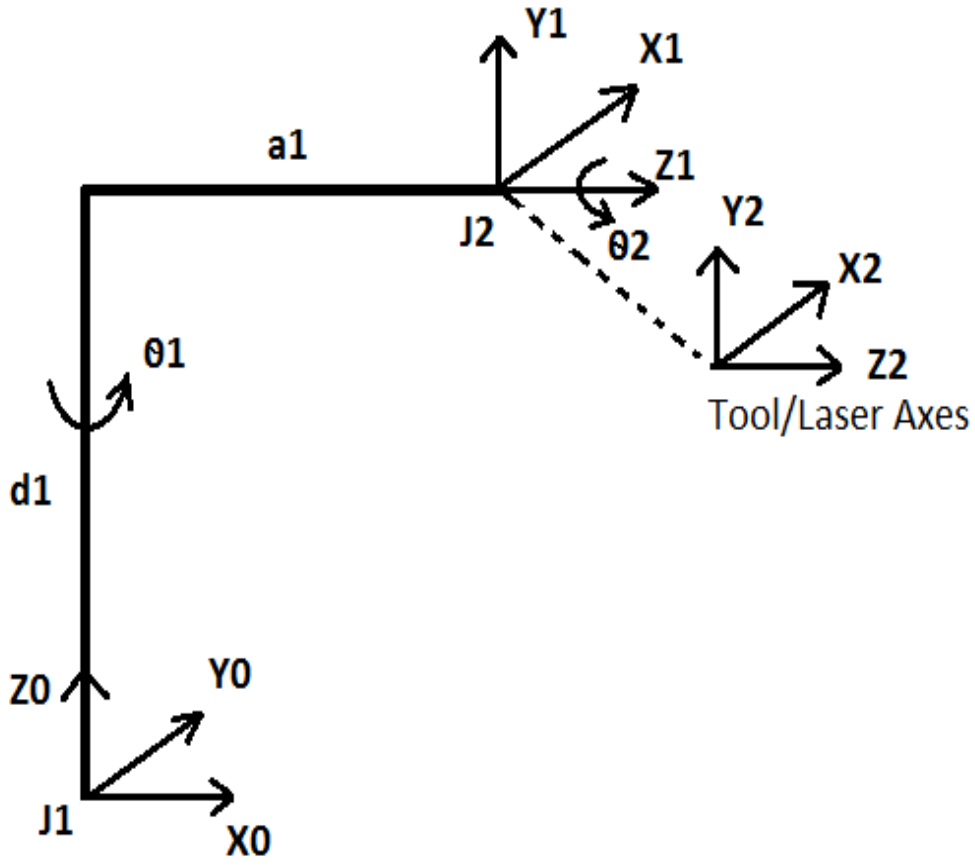


Fig. 12 LCD of 2-axis Robotic Arm

4.1. Algorithms

4.1.1. MATLAB

- Calculate pixels per degree (n) swapped
- Divide the complete frame into segments with n pixels in each segment.
- The number of degrees to be moved by servo motor will be the number of segments to be swapped.
- Thus calculate the number of segment to be swapped and convert it to corresponding angle value.

4.1.2. Arduino

- Arduino receives the pulse width for all the servos from MATLAB through Serial port.

- If the received pulse is appended by alphabet ‘b’ it gives pulse to the base motor.
- Similarly, if ‘s’ then shoulder motor and ‘f’ for frame switching.
- Thus the laser pointer acquires the target based on these pulse values.

4.2. System Flowchart:

The system used to implement this flow chart is as follows:

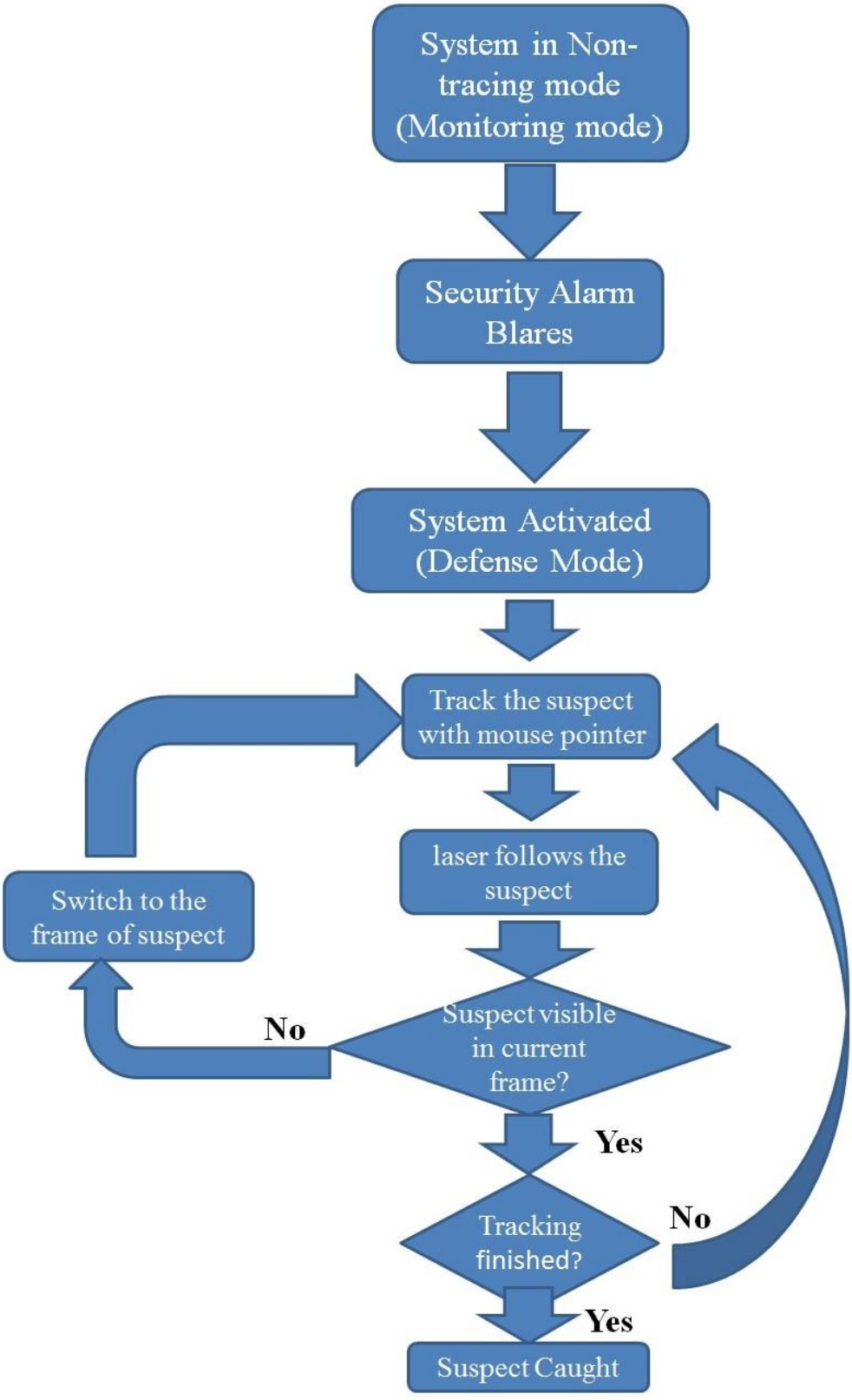


Fig 13 . System Flow Chart

System Block Diagram:

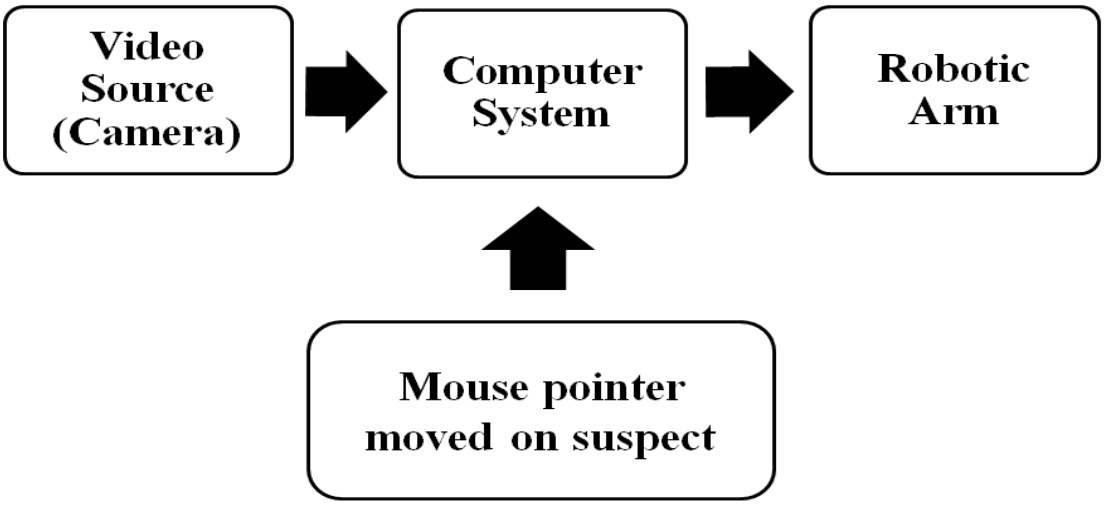


Fig 14. Block diagram

Camera:

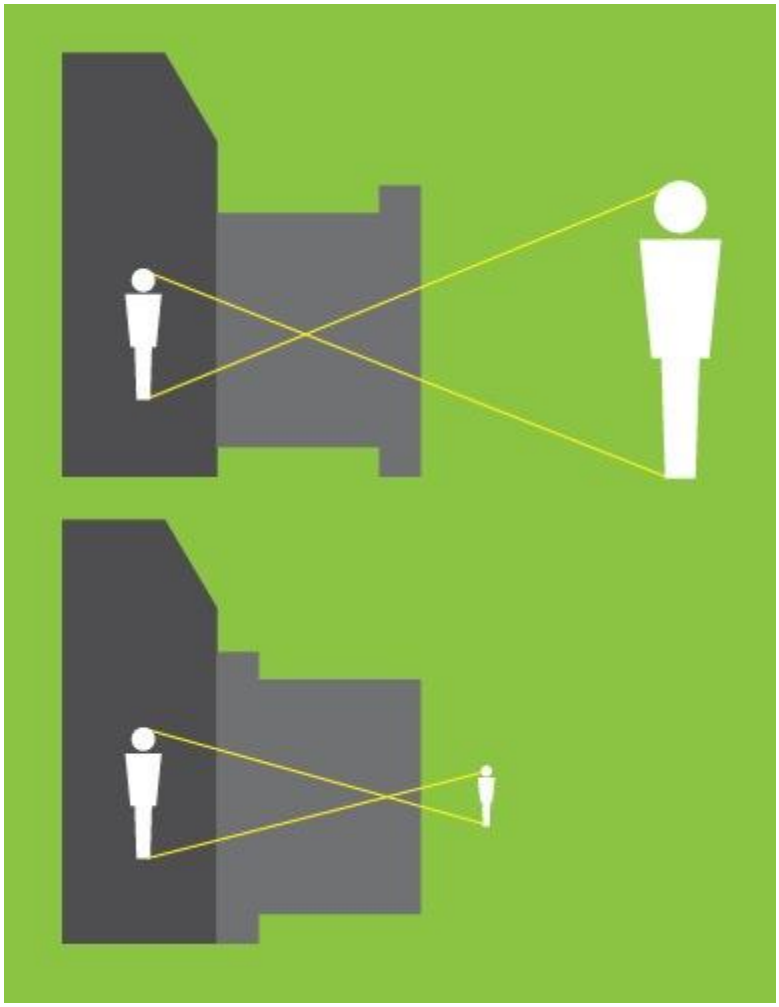


Fig 15. Camera view

- Camera captures the entire scene in an image as shown below.
- This image is read in the MATLAB software.
- No. of pixels of the image are known.
- We know that, in the image the depth of the scene is lost. Hence it becomes difficult to trace the object on the scene.
- To solve this problem, the object is traced on the scene in the image itself so the loss due to absence of the depth information is minimized.

Computer:

- As stated above, the image from the camera is read in the computer using MATLAB software.
- To read the images continuously and extract its coordinates, a GUI is developed as shown below which shows the site to the officer in the surveillance room.
- He can make the system point at a particular point in the scene using the mouse. The GUI reads the coordinates of the mouse and thus of the target point.
- The x and y coordinates of the image from the gui gives the 2-D approximation of the position of the 3-D object in 3-D scene.
- These coordinates are used to move the pan-tilt robotic ARM to track the target.
- To achieve this tracking, the algorithm used is as follows.

This algorithm is written for tracking in X-direction. Tracking in Y-direction is achieved in the same manner.

1. Find the lowest value of pulse to be given to the servo motor to point the laser at lowest coordinate value (P_{hl}).
2. Find the highest value of pulse to be given to the servo motor to point the laser at highest coordinate value (P_{hh}).
3. Subtract the lowest pulse value from the highest pulse value to get the range of operating pulse value (P_{hr}).

$$P_{hr} = P_{hh} + P_{hl}$$

4. Divide this range of pulse by 10 to get the number of degrees in which the entire range of coordinate value is covered(D_h).
5. Similarly calculate the coordinate range.(C_r)
6. Now divide the coordinate range with the number of degrees calculated above and rounded off.
7. This is the number of pixels that will be skipped in every degree of movement.(C_s)

$$C_s = C_r / D_h$$

8. To calculate the position of the servo motor in degrees, the real-time value of coordinate is divided by 'C_s' and rounded off.

$$P_{hd} = X/C_s$$

9. Now the pulse value to be given to the motor is calculated using the formulae.

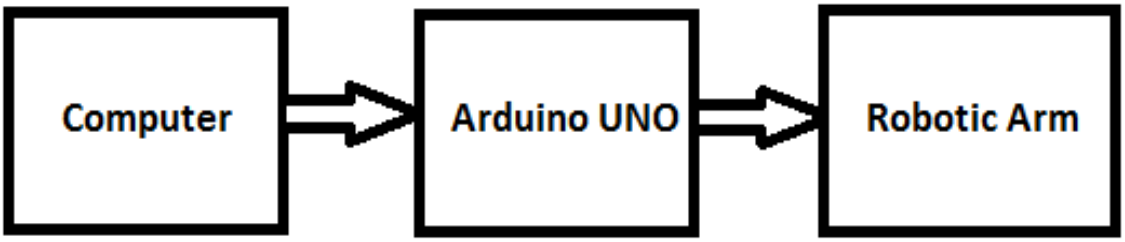
$$P_h = P_{hl} + (P_{hd} * 10)$$

Robotic ARM:

1. This pulse value is calculated for both the motors tracking in x- and y-directions and passed to the motor via arduino.
2. This results in the movement of the motor and thus the ARM will point towards the target.

Chapter 5

IMPLEMENTATION



System Block Diagram

Fig. 16

5.1. Computer:

- For implementing the whole system, first of all a computer is required. Then MATLAB software is installed on computer. MATLAB (**mat**rix **lab**oratory) is a multi-paradigm numerical computing environment and fourth-generation programming language developed by MathWorks. Latest version of MATLAB is 8.3(R2014a).
- By using the image acquisition toolbox in the MATLAB the video is fetched from the camera.
- Then program to fetch the mouse pointer location (2D coordinates) from the video is written in MATLAB. These coordinates are then transmitted serially to Arduino UNO block.
- A GUI (Graphic User Interface) is developed to implement user friendly functions such as start and stop.
- Another software required to be installed on computer is Arduino IDE.
- Program to serially receive the mouse pointer locations is written in the sketch of Arduino IDE. This program serially reads the coordinates transmitted by the MATLAB. Also the programming of the robotic arm is implemented in the same program which is discussed in the later part.
- The Arduino UNO is connected to computer through a USB cable at any one of the computer's USB port. Programmer must know the port number to avoid any problem in serial communication between the computer and Arduino UNO.

5.2.Arduino UNO:

- The program written in the sketch of Arduino IDE is then verified and uploaded into Arduino UNO.
- Also there already exists a library in the Arduino IDE for generation of PWM signal of variable pulse width with a single instruction which makes programmer's job simple.

Fig. 17 Arduino UNO with PWM pins highlighted in red

- These pins are used to apply PWM control signals to the servo motors attached in the robotic arm.
- Hence the control signal of the servo motors is connected to these pins.
- The successful serial communication is indicated by two LEDs TX and RX connected on board.

5.3. ROBOTIC ARM:

- For construction of chassis of robotic arm the following aluminium part is required.

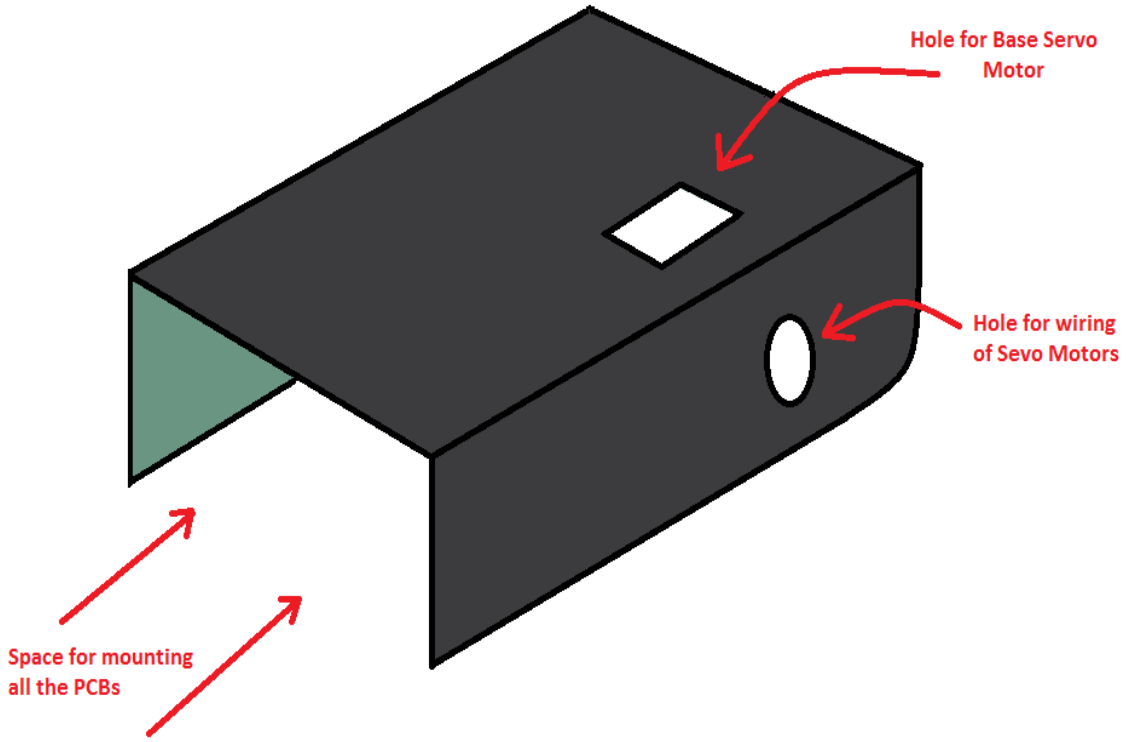


Fig. 18 Chassis

- This chassis of the robot serves following 4 purposes:
    1. First is to mount the servo motors.
    2. To mount the Arduino UNO board.
    3. To mount the camera.
    4. To mount external power supply circuit for the Servo motors.
- Hence the chassis must be big enough to serve the above purposes.
- Power supply to the robotic arm is applied separately. It is because servo motors are low impedance devices and high current flows through its coils. These high currents may damage the components on the Arduino UNO hence a separate power supply is required.
- This separate power supply provides a high current capability and voltage is 6Volts. (IC 7806 is used as voltage regulator)
- The above chassis is drilled to get a rectangular hole in it. The base (rotatory) servo motor is attached on this drilled rectangular hole by using bolts or a strong adhesive.
- Similarly another servo motor is attached to the shaft of the base rotating servo motor by considering the fact that the axes of both motors are perpendicular to each other.

- Hence the base motor serves the movement along the X axis (Horizontal) and the second motor serves the movement along the Y axis (Vertical).
- Then a tracking laser is mounted as the tool of the robotic arm.
- Camera is mounted in such a way that it points the region where tracking is to be performed.
- After mounting the base motor and the camera the robotic arm looks as follows:

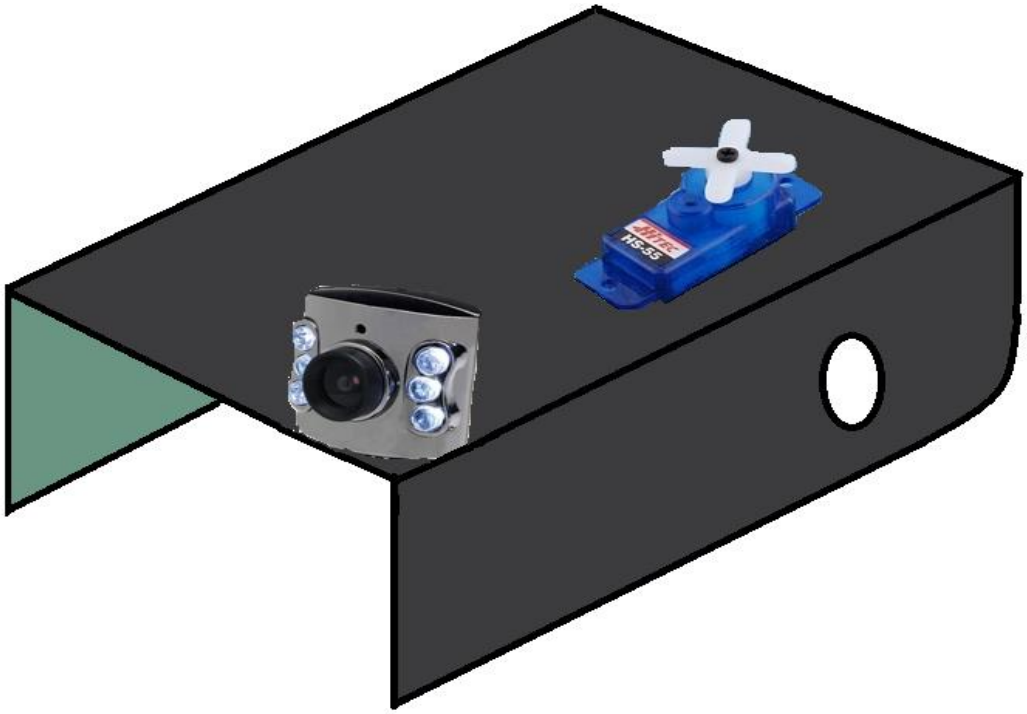


Fig. 19 Chassis Set up

Chapter 6

RESULTS AND DISCUSSION

Fig 20. The robotic arm

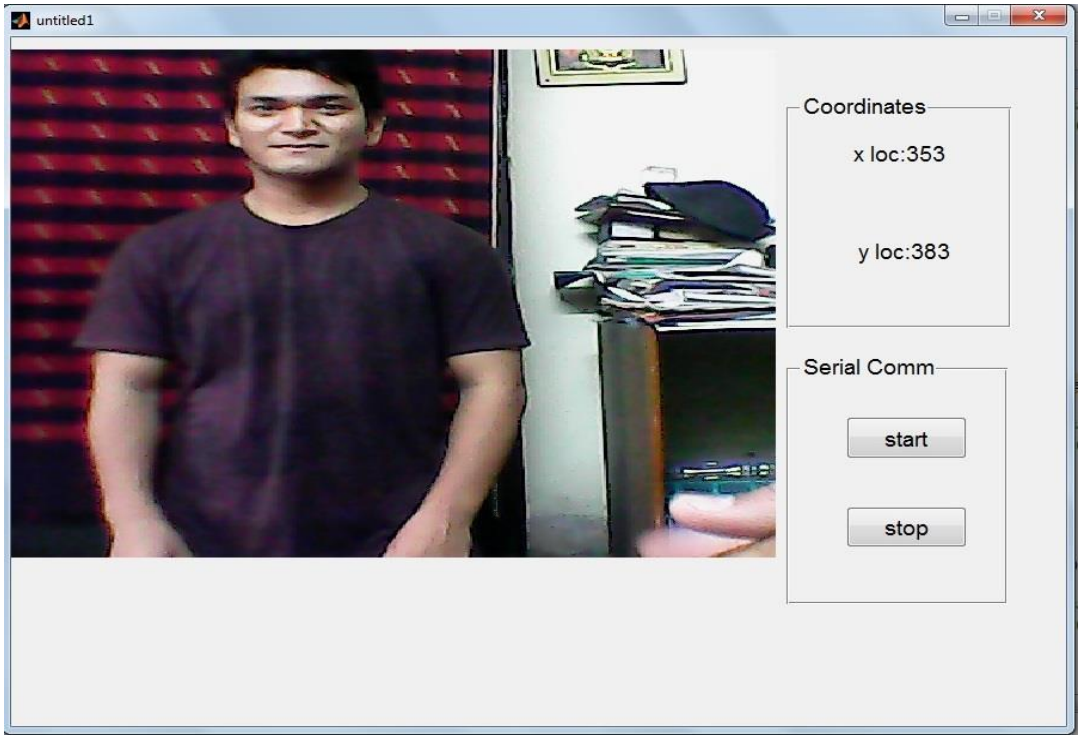


Fig 21. The MATLAB GUI:

- Automated tracking using template matching fails to give accurate results even in moderately complex backgrounds.

- For such real time use of surveillance systems, manual tracking can gives the best results.

Chapter 7

CONCLUSION

- The servo motors used here are with the precision of 1°. This puts a constraint on the distance of the target from the installed system because as this distance increases, the arc traced per step keeps on decreasing.
- Automatic tracking results in noisy output in the complex background and thus hampers the result to a great extent. In such Surveillance system scenario this could prove fatal and undesirable as the final result has to be very précised and accurate.
- This Drawback was overcome by using Manual Tracking.
- Manual Tracking gives a highly accurate and precise output and solves the problem described above.

FUTURE SCOPE

- Here single Camera surveillance system is implemented.
- Single camera system covers only small portion of the room. To implement the system in a bigger room, system with multiple-cameras can be used which increases the area under surveillance.

APPENDIX A

Arduino Program

```
// type servo position 0 to 180 in serial monitor
// or for writeMicroseconds, use a value like 1500
// for IDE 0022 and later
// Powering a servo from the arduino usually *DOES NOT WORK*.

String readString,subString;
#include <Servo.h>
Servo myservob;  // create servo object to control base servo
Servo myservos;  // create servo object to control shoulder servo
Servo myservof;  // create servo object to control frame servo
int laser = 13;  // laser on pin 13
void setup() {
  Serial.begin(9600);
  myservob.writeMicroseconds(1500); //set initial servo position if desired
  myservob.attach(9, 500, 2300);   //the pin for the servo control, and range if desired
  myservos.writeMicroseconds(1500); //set initial servo position if desired
  myservos.attach(10,500, 2300);
  myservof.writeMicroseconds(1300); //set initial servo position if desired
  myservof.attach(11,500, 1900);
  pinMode(laser, OUTPUT);          //pin 13 set as output
  Serial.println("servo-test-22-dual-input"); // so I can keep track of what is loaded
}
void loop() {
  while (Serial.available()) {
    char c = Serial.read();  //gets one byte from serial buffer
    readString += c; //makes the string readString
    delay(2);  //slow looping to allow buffer to fill with next character
  }



  if (readString.length() >0) {
```

```
int n= readString.length();
char p=readString[n-1];
subString = readString.substring(0,n-1) ;
n = subString.toInt();  //convert readString into a number
  // auto select the servo based on appended alphabet
  if(p == 'b')
  {
    Serial.print("writing BaseAngle: ");
    Serial.println(n);
    myservob.writeMicroseconds(n);
  }else if(p == 's')
  {
    Serial.print("writing ShoulderAngle: ");
    Serial.println(n);
    myservos.writeMicroseconds(n);
  }else if(p == 'f')
  {
    Serial.print("writing FrameAngle: ");
    Serial.println(n);
    myservof.writeMicroseconds(n);
  }
  //Switching laser
  if(readString == "ON")
  {
    Serial.print("Turning on laser ");
    digitalWrite(laser, HIGH); //turn on laser
  }else if(readString == "OFF")
  {
    Serial.print("Turning off laser ");
    digitalWrite(laser, LOW); //turn off laser
  }
  readString=""; //empty for next input
} }
```

MATLAB Code

GUI program:

```
function varargout = untitled1(varargin)
% UNTITLED1 MATLAB code for untitled1.fig
%      UNTITLED1, by itself, creates a new UNTITLED1 or raises the existing
%      singleton*.
%
%      H = UNTITLED1 returns the handle to a new UNTITLED1 or the handle to
%      the existing singleton*.
%
%      UNTITLED1('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in UNTITLED1.M with the given input arguments.
%
%      UNTITLED1('Property','Value',...) creates a new UNTITLED1 or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before untitled1_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to untitled1_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help untitled1

% Last Modified by GUIDE v2.5 04-Apr-2014 23:34:35

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @untitled1_OpeningFcn, ...
                   'gui_OutputFcn',  @untitled1_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before untitled1 is made visible.
```

```
function untitled1_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to untitled1 (see VARARGIN)

% Choose default command line output for untitled1

% Creating video object and displaying it on GUI axes
handles.output = hObject;
axes(handles.axes1);
vid=videoinput('winvideo',2,'YUY2_640x480');
hImage=image(zeros(640,480,3),'Parent',handles.axes1);
preview(vid,hImage);

global on_off_flag; % variable for laser on_off control
on_off_flag = 0;    % Initially off
% Creating serial port object
global s;
s = serial('COM6');

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes untitled1 wait for user response (see UIRESUME)
% uiwait(handles.fig_mouse);


% --- Outputs from this function are returned to the command line.
function varargout = untitled1_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on mouse motion over figure - except title and menu.
function fig_mouse_WindowButtonMotionFcn(hObject, eventdata, handles)
% hObject    handle to fig_mouse (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

pos = get(hObject, 'currentpoint'); % get mouse location on figure
global s;
global x;
global y;
```

```
x = pos(1); y = pos(2); % assign locations to x and y


set(handles.lbl_x, 'string', ['x loc:' num2str(x)]); % update text for x loc
set(handles.lbl_y, 'string', ['y loc:' num2str(y)]); % update text for y loc

% Calculating Base and Shoulder Angles
flag = strcmp(s.status , 'open'); % returns 1 if both strings are equal
if flag == 1

a1 = x/16;
a1 = round(a1) + 1;
pulse1 = 1640 + 100 - a1*10;
pulstr1 = num2str(pulse1);
fwrite(s,strcat(pulstr1,'b'));

a2 = y/16;
a2 = round(a2) + 1;
pulse2 = 1640 + 200 - a2*10;
pulstr2 = num2str(pulse2);
fwrite(s,strcat(pulstr1,'b'));
fwrite(s,strcat(pulstr2,'s'));
end

% --- Executes on button press in start.
function start_Callback(hObject, eventdata, handles)
% hObject    handle to start (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global s;
fopen(s);   % Opening serial port

% --- Executes on button press in stop.
function stop_Callback(hObject, eventdata, handles)
% hObject    handle to stop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global s;
fclose(s);  % Closing serial port


% --- Executes on mouse press over figure background, over a disabled or
% --- inactive control, or over an axes background.
function fig_mouse_WindowButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to fig_mouse (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Toggling Laser on each click
global s;
global on_off_flag;
if on_off_flag == 1
   fwrite(s,'OFF');
   on_off_flag = 0;

elseif on_off_flag == 0
   fwrite(s,'ON');
   on_off_flag = 1;
end


% --- Executes on key press with focus on fig_mouse and none of its controls.
function fig_mouse_KeyPressFcn(~, eventdata, handles)
% hObject    handle to fig_mouse (see GCBO)
% eventdata  structure with the following fields (see FIGURE)
%   Key: name of the key that was pressed, in lower case
%   Character: character interpretation of the key(s) that was pressed
%   Modifier: name(s) of the modifier key(s) (i.e., control, shift) pressed
% handles    structure with handles and user data (see GUIDATA)
disp('Switching to frame :');
disp(eventdata.Key)
global s;
switch eventdata.Key
   case '1'
      fwrite(s,'600f');

   case '2'
      fwrite(s,'900f');

   case '3'
      fwrite(s,'1300f');

end
```

Main program:

```
global x;
global y;
global s;


while(1)
   a1 = x/16;
   a1 = round(a1) + 1;
   pulse1 = 1640 + 100 - a1*10;
   pulstr1 = num2str(pulse1);
   fwrite(s,strcat(pulstr1,'b'));
```

```
    a2 = y/16;
    a2 = round(a2) + 1;
    pulse2 = 1640 + 200 - a2*10;
    pulstr2 = num2str(pulse2);
    pause on;
    pause(0.1);
    fwrite(s,strcat(pulstr1,'b'));
    fwrite(s,strcat(pulstr2,'s'));

end;
```

APPENDIX B

Servo Specification

Futaba S3003:

Detailed Specifications			
Control System:	+Pulse Width Control 1520usec Neutral	Current Drain (4.8V):	7.2mA/idle
Required Pulse:	3-5 Volt Peak to Peak Square Wave	Current Drain (6.0V):	8mA/idle
Operating Voltage:	4.8-6.0 Volts	Direction:	Counter Clockwise/Pulse Traveling 1520-1900usec
Operating Temperature Range:	-20 to +60 Degree C	Motor Type:	3 Pole Ferrite
Operating Speed (4.8V):	0.23sec/60 degrees at no load	Potentiometer Drive:	Indirect Drive
Operating Speed (6.0V):	0.19sec/60 degrees at no load	Bearing Type:	Plastic Bearing
Stall Torque (4.8V):	44 oz/in. (3.2kg.cm)	Gear Type:	All Nylon Gears
Stall Torque (6.0V):	56.8 oz/in. (4.1kg.cm)	Connector Wire Length:	12"
Operating Angle:	45 Deg. one side pulse traveling 400usec	Dimensions:	1.6" x 0.8"x 1.4" (41 x 20 x 36mm)
360 Modifiable:	Yes	Weight:	1.3oz. (37.2g)

REFERENCES

[1] AlperYilmaz, Omar Javed and Mubarak Shah,"Object Tracking: A Survey",ACM Computing Surveys Vol.38,No.4,Article 13,December 2006.

[2] A. J. Lipton, H. Fujiyoshi, and R.S. Patil. "Moving target classification and tracking from real-time video". In Proc. of Workshop Applications of Computer Vision, pages 129–136, 1998.

[3] A. M. McIvor." Background subtraction techniques". In Proc. of Image and Vision Computing, Auckland, New Zealand, 2000.

[4] Birchfield, S. 1998. "Elliptical head tracking using intensity gradients and color histograms." In IEEE Conference on Computer Vision and Pattern Recognition (CVPR).232–237.

[5] Comanicu, D., Ramesh V., Andmeer, P. 2003. "Kernel-based object tracking" IEEE Trans. Patt. Analy. Mach.Intell. 25, 564–575.

[6] C. Papageorgiou, T. Evgeniou, and T. Poggio. A trainable pedestrian detection system. In Proc. of IEEE Int. Conf. on Intelligent Vehicles, pages 241–246, Germany, October 1998.

[7] http://cegopa.blogspot.com.es/2012/08/l298n-arduino-mega-2560-stepper-em-257.html

[8] https://code.google.com/p/stepperserialcontrol/downloads/list , stepper motor.

[9] http://en.nanotec.com/support/tutorials/stepper-motor-and-bldc-motor-animation/, steppermotor

[10] http://www.mathworks.in/matlabcentral/fileexchange/authors/92623 , red object tracking

[11] http://www.mathworks.in/help/images/ref/normxcorr2.html#bqkhmom-1

[12] http://www.youtube.com/watch?v=QOzmDen4HU&list=TLO_Pcb-g19o8 , templatematching

[13] https://www.youtube.com/watch?v=BfMfysmfoNM

[14] https://www.youtube.com/watch?v=kLd_JyvKV4Y

[15] https://www.youtube.com/watch?v=mVx02s1fHIY&list=PLA2F3F900711B07C3

[16] https://www.youtube.com/watch?v=CrEW89RrtB4&list=PLA2F3F900711B07C3

[17] https://www.youtube.com/watch?v=RP5VeeTceeo

[18] https://www.youtube.com/watch?v=cKf-0j6Ju90

[19] J. Heikkila and O. Silven." A real-time system for monitoring of cyclists and pedestrians". In Proc. of Second IEEE Workshop on Visual Surveillance,pages 74–81, Fort Collins, Colorado, June 1999.

[20] J.K. Aggarwal and Q. Cai.,"Human motion analysis: A review. Computer Vision and Image Understanding", 73(3):428–440, March 1999.

[21] P. KaewTraKulPong and R. Bowden. "An Improved Adaptive Background Mixture Model for Real-time Tracking with Shadow Detection", chapter 11,pages 135–144. Video-Based Surveillance Systems. Kluwer Academic Publishers,Boston, 2002.

[22] R. T. Collins et al. "A system for video surveillance and monitoring: VSAM final report". Technical report CMU-RI-TR-00-12, Robotics Institute, Carnegie Mellon University, May 2000.

[23] T. Brodsky et al. "Visual Surveillance in Retail Stores and in the Home", chapter 4, pages 51–61. Video-Based Surveillance Systems. Kluwer Academic Publishers, Boston, 2002.

[24] Yigithan Dedoglu,"Moving Object Detection, Tracking and Classification for Smart Video Surveillance",Thesis,August 2004

****