T.C.

MARMARA UNIVERSITY

INSTITUTE FOR GRADUATE STUDIES IN

PURE AND APPLIED SCIENCES

BUILDING AN AUTONOMOUS WHEELED TRACKING ROBOT AND LOCATION POSITIONING

Halil DEMİREZEN

(141524120089004)

THESIS

FOR THE DEGREE OF MASTER OF SCIENCE

IN

COMPUTER ENGINEERING

SUPERVISOR

Yrd. Doç. Dr. Mehmet Baran

İSTANBUL 2011

ACKNOWLEDGE

I would like to thank to Asst. Prof. Dr. Mehmet Baran for his valuable efforts and supports in realizing this thesis study. I also like to thank to Sevim Songün Demirezen for supporting me throughout this work and for her patience as well. Finally, to my little precious, Kerem.

CONTENTS

ÖZET

OTONOM TEKERLEKLİ TAKİP ROBOTU OLUŞTURMA VE KONUMLANDIRMA

Bu tez kapsamında, otomom takip robotu gerçekleştirilmiştir. Robot, birbirine bağımlı üç bileşenden oluşmaktadır: video kamera, işleme birimi, şase ve hareket bölümleri.

"Logitech™ Quickcam Pro for Notebook" video kamera kaynak aygıt olarak kullanılmıştır. Kameranın Linux işletim sisteminde çalışması için uvc sürücüsü yüklenmiş ve aygıttan görüntü almak, işlemek için V4L2 Linux çekirdek API arayüzü kullanılmıştır. Takip algoritmaları V4L2 API arayüzü ile alınan görüntü üzerinde uygulanmış ve takip sonuçlarına göre sistem hareket birimlerine takip edilecek nesneye doğru hareket için gerekli komutları göndermiştir.

Takip algoritması takip edilecek nesne olarak kırmızı renkli bir daireyi kullanmıştır. Sistemin çalışması sırasında ilk olarak nesne kameraya yakın bir bölgede tutulmuştur. Böylece robot nesnenin yerini tayin etmiştir. Daha sonra nesne, sağa, sola, ileri hakeret ettirilmiş, hakeret esnasında ise durma işlemi gerçekleştirmiştir. Bu şartlar altında, robot nesneyi kamera görüntüsü üzerinde kullandığı takip penceresinin parametreleri – pencerenin görüntüdeki x-y koordinatları, pencerenin boyutu - yardımıyla takip etmiştir.

Takip penceresi bütün görüntü içerisinde nesnenin yerini işaretlemek için kullanılan mavi renkli bir karedir. Kaynak aygıttan – video kamera – görüntüler geldikçe, görüntü üzerindeki her pikselin RGB renk bileşenleri nesnenin yerini tespit etmek için kullanılmıştır. Gerçekleştirim sırasında, kabul edilebilir kırmızı pikseller (RGB değerleri oranları ile kırmızı olduğu kabul edilen) takip penceresinin yeri, boyunu hesaplamada dikkate alınmıştır.

Hedef nesne belirli bir yöne hareket ettikçe, sistem hareket birimleri nesneyi görüntü üzerinde sabit bir yerde tutmak için hareket birimlerine komutlar gönderir. Örneğin nesne çok ileri gitmişse, bu takip penceresinin küçüldüğü anlamına geliyor, boyutu büyütmek için ileri doğru hareket komutları verilir.

Şubat, 2011 **Halil Demirezen**

ABSTRACT

BUILDING AN AUTONOMOUS WHEELED TRACKING ROBOT AND LOCATION POSITIONING

In the scope of this thesis, autonomous tracking robot has been realized. The robot consists of three dependent components, video streaming camera, processing unit, and chassis and moving parts.

"Logitech™ Quickcam Pro for Notebook" video camera has been implemented as video streaming input device. USB device driver for Linux operating system, named uvc, has been loaded, and V4L2 kernel API interface for video devices on Linux operating system has been used to query frame buffers and process by applying tracking algorithms and depending on the result of tracking operations, chassis including wheels and motors have been instructed to track the target object.

Tracking algorithm has used a red colored circle as target object. The object has been held close to the robot (video device part) at first step to help tracker locate the target object in the frame buffer. Then, the object moved forward, left, right and stopped. Under these circumstances, tracker followed the object depending on the parameters - which are, size, x-y position within the frame buffer - of the tracking window.

Tracking window is a blue painted square on the output screen to locate the target object within a single frame. As the frames are grabbed from the input device, RGB components of each pixel in the frame are used to extract red circle object. To do so, accepted red pixels (whose redness rate is above a predefined threshold value) are taken into consideration to construct new tracking window's width, height and x-y location.

As the target moves in front of the robot, the tracking window will move in the same direction to locate the object. In case of tracking window passes some limit values in height, width or x-y coordinate, algorithm will instruct robot movement motors to satisfy the parameters of the tracking window to reside within the desired values. Thus, autonomous tracking is fulfilled.

February, 2011 **Halil Demirezen**

SYMBOLS

x **:** Location in x coordinate axis

y **:** Location in y coordinate axis

kb/s **:** Kilobits per second

Mb/s **:** Megabits per second

fps **:** Frame per second

id **:** Identification

__u32 **:** Unsigned 32 bit type

__u8 **:** Unsigned 8 bit type

__s64 **:** Signed 64 bit type

int **:** Integer type

char **:** Character type

__u64 **:** Unsigned 64 bit type

Hz **:** Hertz

KHz **:** Kilohertz

struct **:** Structure type

void **:** Not belong to any type

ABBREVIATIONS

GPL **:** General Public License

CPU **:** Central Processing Unit

RAM **:** Random Access Memory

FIFO **:** First In First Out

GDT **:** Global Descriptor Table

UVC **:** Universal Serial Bus Video Class

USB **:** Universal Serial Bus

API **:** Application Programming Interface

HP **:** Hewlett-Packard

V4L2 **:** Video For Linux 2

OEM **:** Original Equipment Manufacturer

AF **:** Auto Focus

RGB **:** Red Green Blue

ITU **:** International Telecommunication Union

SMTPE **:** Society of Motion Picture and Television Engineers

BCD **:** Binary Coded Decimal

PAL **:** Phase Alternating Line

SECAM **:** Séquentiel Couleur à Mémorie

NTSC **:** National Television System Committee

ISO **:** International Organization for Standardization

AVL **:** Automatic Volume Limiter

SDL **:** Simple DirectMedia Layer

FIGURES

PAGE NO

TABLES

PAGE NO

CHAPTER I

INTRODUCTION AND AIM

I.1. INTRODUCTION

In this thesis, we have constructed a four-wheeled autonomous vehicle whose purpose is to track a specific fiducial object. Our fiducial is a red triangle within a cyan rectangle. We use an off-the-shelf USB webcam as an the input device. The central procesing unit is an Dell Latitude E4300 laptop with Centrino 2 Core Duo CPU. This laptop inputs approximately three frames per second from the webcam and detects the fiducial within every frame it receives. Then, it calculates the distance and orientation of the fiducial with respect to the webcam. It generates commands to drive the vehicle to track the fiducial at a certain distance. Lastly, the motor control units receive commands from the central processing unit and generate commands to drive steering wheel servo and traction servo.

In the light of this abstract, we divide our exposition into four chapters. Each chapter corresponds to a stage in the processing of a frame and this processing is repeated for all captured frames. See Figure I.1.

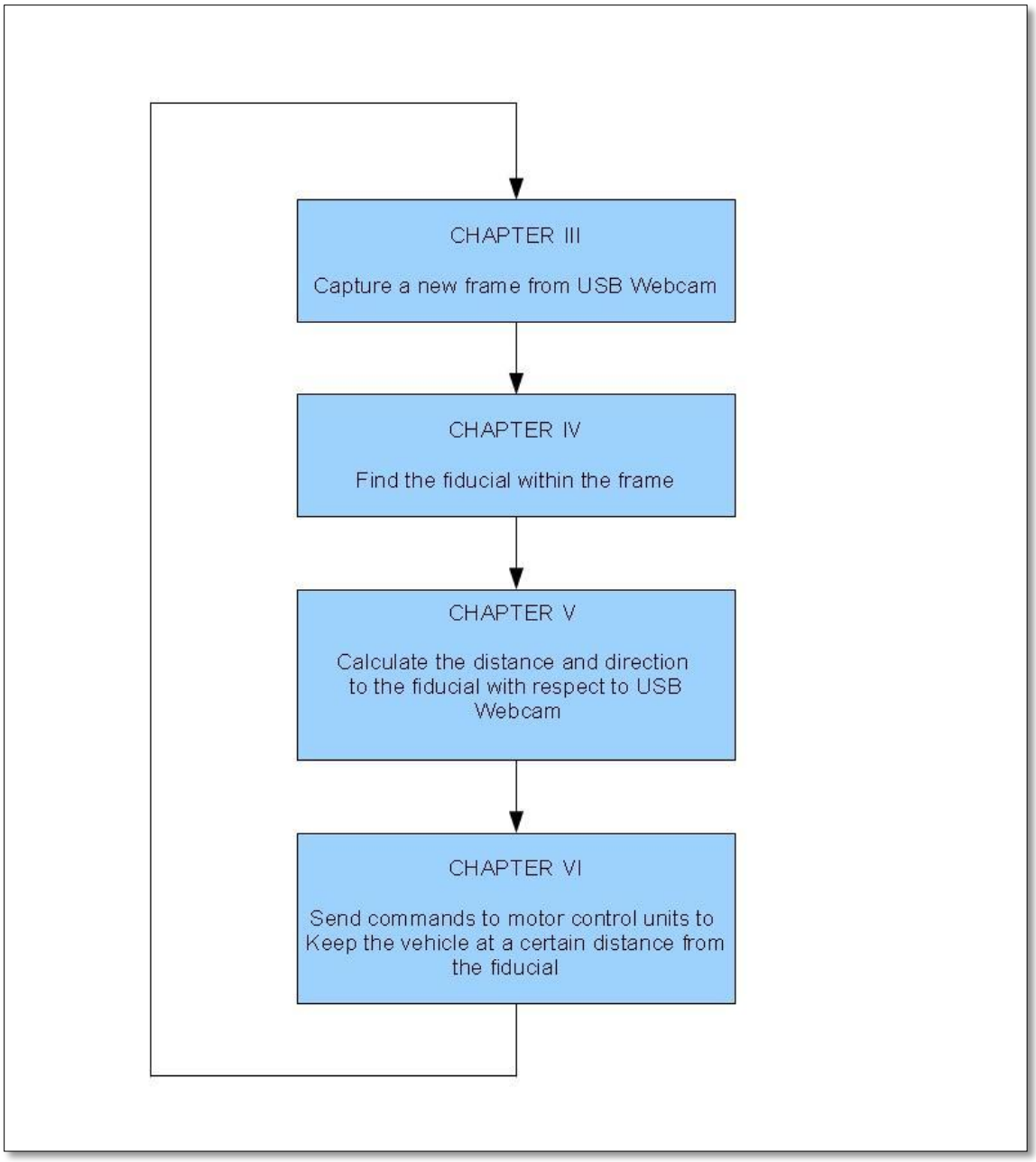


Figure I.1 - Chapters

CHAPTER II

GENERAL INFORMATION

II.1. OBJECT DETECTION

Detecting certain shapes, including triangles, in natural images is a frequently encountered problem in computer vision literature, where the main application is road sign detection [1], [10]. The most frequently employed algorithm in this task is Fast Radial Symmetry Detector [2] of Gareth et al. This algorithm is not useful for our purposes, as it only detects equilateral triangles, which are exclusively used for road signs. Also, this algorithm assumes that a car approaches to a traffic sign frontally, so it does not take the perspective deformation into account.

Another approach from learning theory is the Adaboost algorithm of Viola and Jones [3], which is good at detecting fixed shape triangles but not arbitrary ones. Therefore, adaboost is also not very useful for our purposes.

To our knowledge, three different approaches also exist for arbitrary triangle detection: [4] uses neural networks, [5] uses Hough Transformation, and [6], [7] use linear regression.

In our thesis, we use a linear regression for triangle edges whether they form lines or not, after connected pixels are defined as contours. The contours are specified as a result of a color transition state machine which looks for red-cyan, cyan-red transition pixels by looking at their color information in HSV color space.

CHAPTER III

CAMERA SUBSYSTEM

III.1. UNDERLYING HARDWARE

Video Camera subsystem consists of a web cam video camera which is connected to the processing unit through usb port. The camera used in the work is Logitech Pro 4000 and the processing unit we used in the thesis is Dell Latitude E4300 wih 2Ghz Intel Processor. Processing unit runs Debian Linux operating system and camera driver for image capturing.

Figure III.1 – Logitech Web Camera

III.2. V4L2 IMAGE CAPTURING LAYER

III.2.1. INTRODUCTION

Video for Linux 2 (V4L2) is a set of APIs for handling video devices on Linux. V4L2 is a replacement for Video for Linux that comes with the kernel by default.

V4L2 is a suite of related driver specifications for different types of video devices and video data. Depending upon the device type, read(), write() operations as well as ioctl operations change characteristically. V4L2 devices are all Linux character-oriented devices with a major number 81. Major number points to the device as we have mentioned previously in Linux section. V4L2 devices are located under /dev directory in Linux file system as well as other devices.

Table III.1 is a list of V4L2 devices and their related character-oriented device file names on the system. We have used /dev/video0 to be our input device. Numerous video devices can be accessed such as /dev/video0, /dev/video1, /dev/video2, and so on.

Table III.2 is a categorized list of V4L2 function calls and related structures.

Device Name	Type Of Device
/dev/video*	Video capture interface
/dev/vfx*	Video effects interface
/dev/codec*	Video codec interface
/dev/vout*	Video output interface
/dev/radio*	AM/FM radio devices
/dev/vtx*	Teletext interface chips
/dev/vbi*	Data services interface

Table III.1: V4L2 Devices

Category	Class	IOCTL	Structure
Common V4L2 Elements	Query Capabilities	VIDIOC_QUERYCAP	v4l2_compatibility
	Memory-Mapping Device Buffers	VIDIOC_REQBUFS	v4l2_requestbuffers
		VIDIOC_QUERYBUF	v4l2_buffer
	Controls	VIDIOC_QUERYCTRL	v4l2_queryctrl
		VIDIOC_QUERYMENU	v4l2_querymenu
		VIDIOC_S_CTRL	v4l2_control
		VIDIOC_G_CTRL	v4l2_control
	Device Performances	VIDIOC_G_PERF	v4l2_performance
Video V4L2 Elements	Enumerating Supported Image Formats	VIDIOC_ENUM_PIXFMT	v4l2_fmtdsc
		VIDIOC_ENUM_FBUFFMT	v4l2_fmtdsc
	Capture Image Format	VIDIOC_G_FMT	v4l2_format
		VIDIOC_S_FMT	v4l2_format
	Frame Buffers	VIDIOC_G_FBUF	v4l2_framebuffer

		VIDIOC_S_FBUF	v4l2_framebuffer
		VIDIOC_G_WIN	V4l2_window
		VIDIOC_S_WIN	v4l2_window
		VIDIOC_PREVIEW	
	Stream Capturing	VIDIOC_QBUF	v4l2_buffer
		VIDIOC_DQBUF	v4l2_buffer
		VIDIOC_STREAMON	
		VIDIOC_STREAMOFF	
	Capture Parameters	VIDIOC_G_PARM	V4l2_streamparm
		VIDIOC_S_PARM	V4l2_streamparm
	Video Inputs	VIDIOC_G_INPUT	V4l2_input
		VIDIOC_S_INPUT	V4l2_input
		VIDIOC_ENUMINPUT	V4l2_input
System Calls and Structures Not used	Tuning	VIDIOC_G_TUNER	V4l2_tuner
		VIDIOC_S_TUNER	V4l2_tuner
		VIDIOC_G_FREQUENCY	V4l2_frequency
		VIDIOC_S_FREQUENCY	V4l2_frequency
	Video Standards	VIDIOC_G_STD	V4l2_standard
		VIDIOC_S_STD	V4l2_standard
		VIDIOC_ENUMSTD	V4l2_standard
	Compressed Capture Parameters	VIDIOC_G_COMP	V4l2_compression
		VIDIOC_S_COMP	V4l2_compression
	Digital Zoom	VIDIOC_ZOOMCAP	V4l2_zoomcap
		VIDIOC_G_ZOOM	V4l2_zoom
		VIDIOC_S_ZOOM	V4l2_zoom
	Reading Captured Images		

Table III.2: IOCTLs and Structures

III.2.2. VIDEO CAPTURING ARCHITECTURE

Since the V4L2 detail is out of scope of this thesis, we use V4L2 for camera initialization and frame grabbing operations. In the first section of this chapter, we have listed some functions and structures as a entry for further investigation of the framework.

In Figure III.2, we have give an overall structure we have used as video device control and data transfer. V4L2_capability structure is used to check if video device is capable of video capturing options. V4L2_format structure is used to check if video device is able to handle requested frame width, height and format, which is YUYV. V4L2_requestbuffers structure is used to see if buffer request is available for the device. And finally, V4L2_buffer is actual entry point to our buffers. The number of buffers

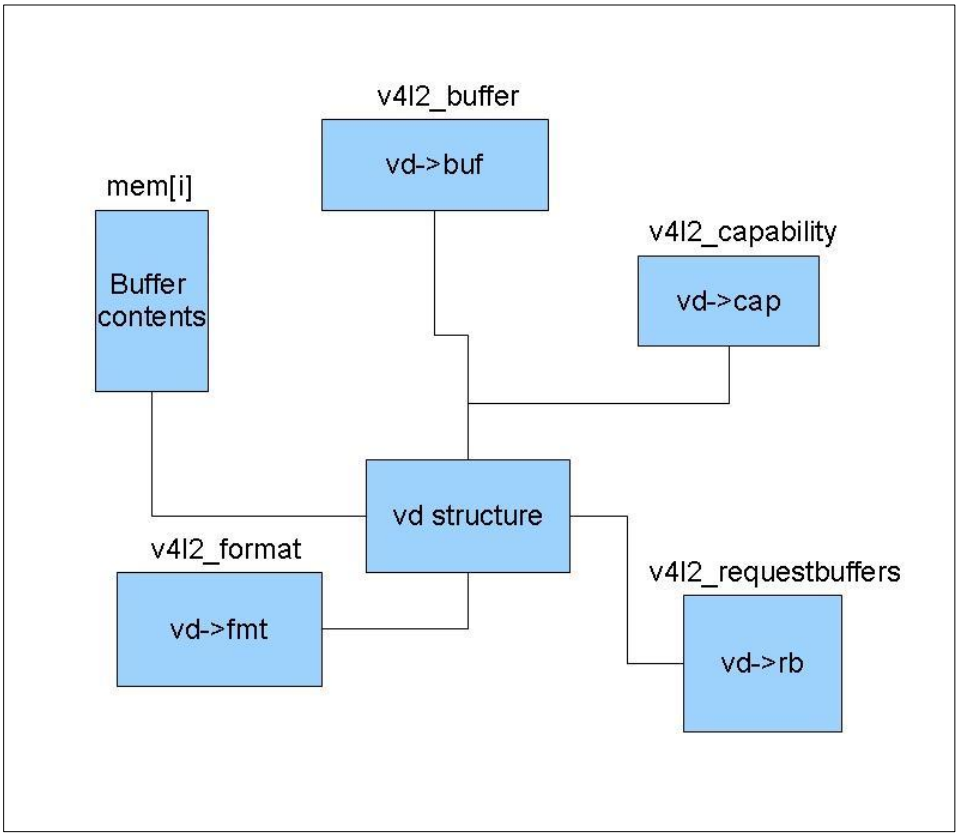


Figure III.2 – vd structure and related structures

can be set depending on the hardware processing capabilities. The buffers are used by the driver which continously fills the available buffers. When the camera is initialized,

the allocated buffers are marked available by queueing call (QBUF). To say, queued buffer is available buffer for camera driver to fill captured frame from the hardware. The buffer which holds image frame in it should explicitly be dequeued and copied into application buffer, in our case vd->framebuffer. As the frame copied into vd->framebufer, the dequeued buffer should be queued for driver use. In Figure III.3 this process is illustrated.

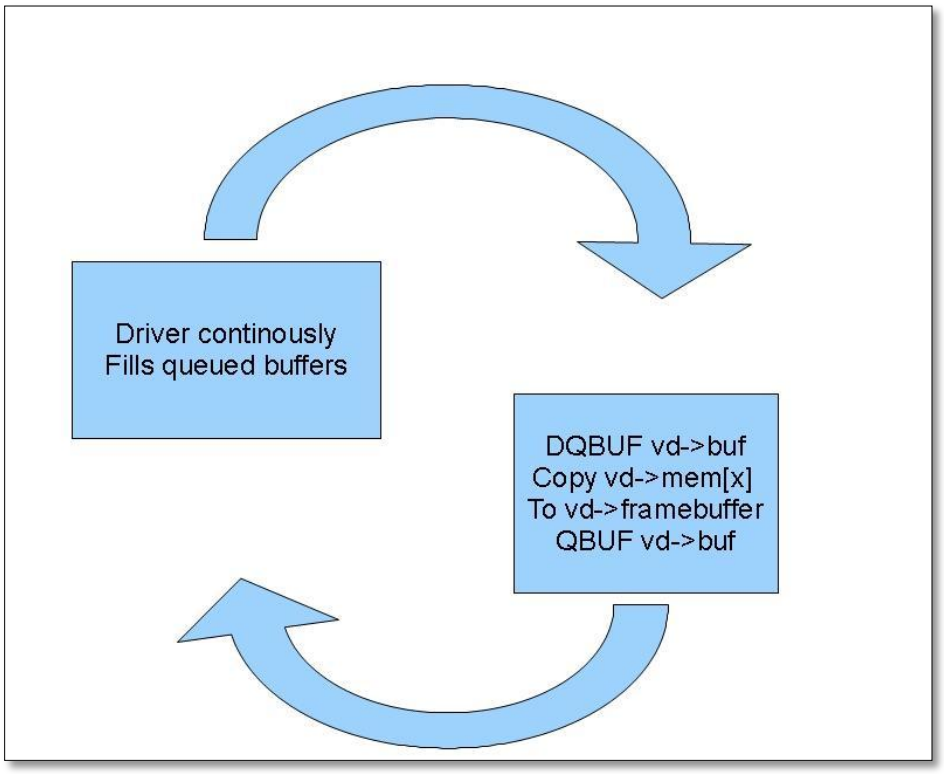


Figure III.3 Frame Grabbing

The vd->framebuffer data is in YUYV format [14]. For our use, we convert YUYV format data to RGB format into output_ptr buffer in the program.

III.2.3. VIDEO CAPTURING ALGORITHMS

In this section, we will detail in the video capturing algorithms which are used in the scope of this thesis in C under Linux environments using V4L2 framework.

The init_video function creates and fill vd structure in vdIn format.

```
int init_video(struct vdIn *vd, char *device, int width, int height,
int fps, int format, int grabmethod, char *avifile)
{
      int ret = -1;
      int i;

      if(vd == NULL ||device==NULL)
            return -1;

      if(width == 0 ||height == 0)
            return -1;

      vd->videodevice = NULL;
      vd->status = NULL;
      vd->pictName = NULL;
      vd->videodevice = (char *)calloc(1, 16 * sizeof(char));
      vd->status = (char *)calloc(1, 100 * sizeof(char));
      vd->pictName = (char *)calloc(1, 80 * sizeof(char));
      snprintf(vd->videodevice, 12, "%s", device);
      fprintf(stdout, "Video: %s\n", vd->videodevice);
      vd->recordtime = 0;
      vd->framecount = 0;
      vd->signalquit = 1;
```

Width and height of screen in pixels.

```
      vd->width = width;
      vd->height = height;
```

The number of frames per second that the devices is requested to capture.

```
      vd->fps = fps;
      vd->formatIn = format;
      vd->grabmethod = 1;
      vd->fileCounter = 0;
      vd->rawFrameCapture = 0;
      vd->rfsBytesWritten = 0;
      vd->rfsFramesWritten = 0;
      vd->captureFile = NULL;
      vd->bytesWritten = 0;
      vd->framesWritten = 0;
```

Calling low level v4l2 initialization. Checking if it is failed.

```
	if(init_v4l2(vd) < 0) {
		printf("Init V4l2 failed !! exit fatal \n");
	}
```

We are creating framebuffer to save captured frame in YUY2 format.

```
	vd->framesizeIn = (vd->width * vd->height << 1);
	vd->framebuffer = (unsigned char *)calloc(1,
			  (size_t)vd->framesizeIn);
	if(!vd->framebuffer)
		goto error;
	return 0;

error:
	free(vd->videodevice);
	free(vd->status);
	free(vd->pictName);
	close(vd->fd);
	return -1;

}
```

Low level v4l2 function which is used by the function above (init_video)

```
static int init_v4l2(struct vdIn *vd)
{
	int i;
	int ret = 0;
```

We are instructing operating system to open device file under /dev directory which is expected to be "/dev/video0"

```
	if((vd->fd = open(vd->videodevice, O_RDWR)) == -1){
		fprintf(stderr, "ERROR OPENIN V4L Interface\n");
		exit(1);
	}
```

Instructing operating system driver to get video camera capabilities.

```
	memset(&vd->cap, 0, sizeof(struct v4l2_capability));
	ret = ioctl(vd->fd, VIDIOC_QUERYCAP, &vd->cap);

	if(ret < 0){
		fprintf(stderr, "Error opening device %s: unable to query
device .\n", vd->videodevice);
		goto fatal;
	}
```

If video device is capable of Video Capture, we will continue to initialize process else we will report an error and quit.

```
      if((vd->cap.capabilities & V4L2_CAP_VIDEO_CAPTURE) == 0){
            fprintf(stderr, "Error opening device %s: video capture
not supported.\n", vd->videodevice);
            goto fatal;
      }
```

Below is checking again the capability structure and streaming feature of the device. If not reporting an error and quiting.

```
      if(vd->grabmethod){
            if(!(vd->cap.capabilities & V4L2_CAP_STREAMING)){
                  fprintf(stderr, "Error opening device %s: doesn't
support streaming i/o.\n", vd->videodevice);
                  goto fatal;
            }
      } else {
            if(!(vd->cap.capabilities & V4L2_CAP_READWRITE)){
                  fprintf(stderr, "Error opening device %s: doesn't
support read i/o.\n", vd->videodevice);
                  goto fatal;
            }
      }
```

Now setting the width and height of frame to be captured in low level function. Pixel format is also set below.

```
      memset(&vd->fmt, 0, sizeof(struct v4l2_format));
      vd->fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
      vd->fmt.fmt.pix.width = vd->width;
      vd->fmt.fmt.pix.height = vd->height;
      vd->fmt.fmt.pix.pixelformat = vd->formatIn;
      vd->fmt.fmt.pix.field = V4L2_FIELD_ANY;
```

After setting the structure with the required information, we initiate a call to the driver to set format with VIDIOC_S_FMT.

```
      ret = ioctl(vd->fd, VIDIOC_S_FMT, &vd->fmt);

      if(ret < 0){
            fprintf(stderr, "Error opening device %s: unable to set
format: %d.\n", vd->videodevice, errno);
            goto fatal;
      }

      if((vd->fmt.fmt.pix.width != vd->width) ||
         (vd->fmt.fmt.pix.height != vd->height)){
            fprintf(stderr, "Error opening device %s: format asked
unavailable get width %d height %d.\n", vd->videodevice, vd-
>fmt.fmt.pix.width, vd->fmt.fmt.pix.height);
            vd->width = vd->fmt.fmt.pix.width;
            vd->height = vd->fmt.fmt.pix.height;
      }
```

```
        struct v4l2_streamparm *setfps;

        setfps = (struct v4l2_streamparm *)
                    calloc(1, sizeof(struct v4l2_streamparm));
        memset(setfps, 0, sizeof(struct v4l2_streamparm));
        setfps->type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        setfps->parm.capture.timeperframe.numerator = 1;
        setfps->parm.capture.timeperframe.denominator = vd->fps;
        ret = ioctl(vd->fd, VIDIOC_S_PARM, setfps);

        memset(&vd->rb, 0, sizeof(struct v4l2_requestbuffers));
        vd->rb.count = NB_BUFFER;
        vd->rb.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        vd->rb.memory = V4L2_MEMORY_MMAP;

        ret = ioctl(vd->fd, VIDIOC_REQBUFS, &vd->rb);

        if(ret < 0){
              fprintf(stderr,
                      "Error opening device %s: unable to allocate
              buffers: %d.\n", vd->videodevice, errno);
              goto fatal;
        }

        for(i = 0; i < NB_BUFFER; i++){
              memset(&vd->buf, 0, sizeof(struct v4l2_buffer));
              vd->buf.index = i;
              vd->buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
              vd->buf.memory = V4L2_MEMORY_MMAP;
              ret = ioctl(vd->fd, VIDIOC_QUERYBUF, &vd->buf);
              if(ret < 0){
                    fprintf(stderr,"Unable to query buffer.\n", errno);
                    goto fatal;
              }
              vd->mem[i] = mmap(0, vd->buf.length, PROT_READ,
MAP_SHARED, vd->fd, vd->buf.m.offset);
              if(vd->mem[i] == MAP_FAILED){
                    fprintf(stderr, "Unable to map buffer.\n", errno);
                    goto fatal;
              }
        }

        for(i = 0; i < NB_BUFFER; i++){
              memset(&vd->buf, 0, sizeof(struct v4l2_buffer));
              vd->buf.index = i;
              vd->buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
              vd->buf.memory = V4L2_MEMORY_MMAP;
              ret = ioctl(vd->fd, VIDIOC_QBUF, &vd->buf);
              if(ret < 0){
                    fprintf(stderr, "Unable to queue buffer (%d).\n",
                    errno);
                    goto fatal;
              }
        }
        return 0;
        fatal:
        return -1;
```

```
}

int uvc_grab(struct vdIn *vd)
{
#define HEADERFRAM1 0xaf
      int ret;

      memset(&vd->buf, 0, sizeof(struct v4l2_buffer));
      vd->buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
      vd->buf.memory = V4L2_MEMORY_MMAP;
      ret = ioctl(vd->fd, VIDIOC_DQBUF, &vd->buf);
      if(ret < 0)
            goto err;

      if(vd->buf.bytesused > vd->framesizeIn)
            memcpy(vd->framebuffer, vd->mem[vd->buf.index],
(size_t)vd->framesizeIn);
      else
            memcpy(vd->framebuffer, vd->mem[vd->buf.index],
(size_t)vd->buf.bytesused);

      ret = ioctl(vd->fd, VIDIOC_QBUF, &vd->buf);
      return 0;
      err:
            vd->signalquit = 0;
            return -1;
}

int video_enable(struct vdIn *vd)
{
    struct v4l2_control control;
    int type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    int ret;

    ret = ioctl(vd->fd, VIDIOC_STREAMON, &type);
    if (ret < 0) {
        printf("Unable to %s capture: %d.\n", "start", errno);
        return ret;
    }

    vd->isstreaming = 1;
    return 0;
}

int video_disable(struct vdIn *vd)
{
     struct v4l2_control control;
     int type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
     int ret;

     ret = ioctl(vd->fd, VIDIOC_STREAMOFF, &type);
     if(ret < 0){
      printf("Unable to stop capture: %d.\n", errno);
        return  ret;
     }
     vd->isstreaming = 0;
     return 0;
```

```
}


int calib_brightness(struct vdIn *vd, unsigned char *p)
{
      struct v4l2_control control;
      int s, x, y;
      unsigned long weight = 0;
      float av;

      for(x = 0; x < vd->width; x++){
                for(y = 0; y < vd->height; y++){
                        s = (x +(y * vd->width)) * 3;
                  weight += (p[s] + p[s+1] + p[s+2])/3;
            }
      }
      av = weight / (vd->width * vd->height);

      if(av > 110 || av < 90){
            control.id = V4L2_CID_BRIGHTNESS;
            if(ioctl(vd->fd, VIDIOC_G_CTRL, &control) < 0){
                  fprintf(stderr, "get brightness value error\n");
            }
            if(av > 110)
                  control.value--;
            else if(av < 90)
                  control.value++;

            if(ioctl(vd->fd, VIDIOC_S_CTRL, &control) < 0){
                  fprintf(stderr, "set brightness value error\n");
            }
            return 1;
      } else
            return 0;


}
```

CHAPTER IV

TRIANGLE DETECTION ALGORITHM

IV.1. INTRODUCTION

Our aim, as we have defined in the previous sections, is to detect a fiducial, which is in our case a red triangle, within a cyan rectangle. Our algorithm should overcome the following difficulties: reflection from shining surfaces, motion blur, different light conditions, and spurious "triangles". While overcoming the difficulties above, our algorithm should remain simple enough to achive a certain fps limit.

IV.2. ALGORITHM ARCHITECTURE

Our algorithm traces the steps given in Figure IV.1. We will devote one section of this chapter to explain each step in more detail.

IV.2.1. Get a frame in YUYV format from USB webcam.

As we have detailed in Chapter III, we grab frame from webcam using uvc_grab function call. This function just fills vd->framebuffer with captured frame in YUYV format.

IV.2.2. Convert YUYV to RGB

We hold a RGB version of captured image for display purpose only. As we will discuss in later section, we run detection algorithm in HSV format rather than RGB format. When we want to draw a line, paint a pixel or print characters to the screen, we use this RGB buffer.

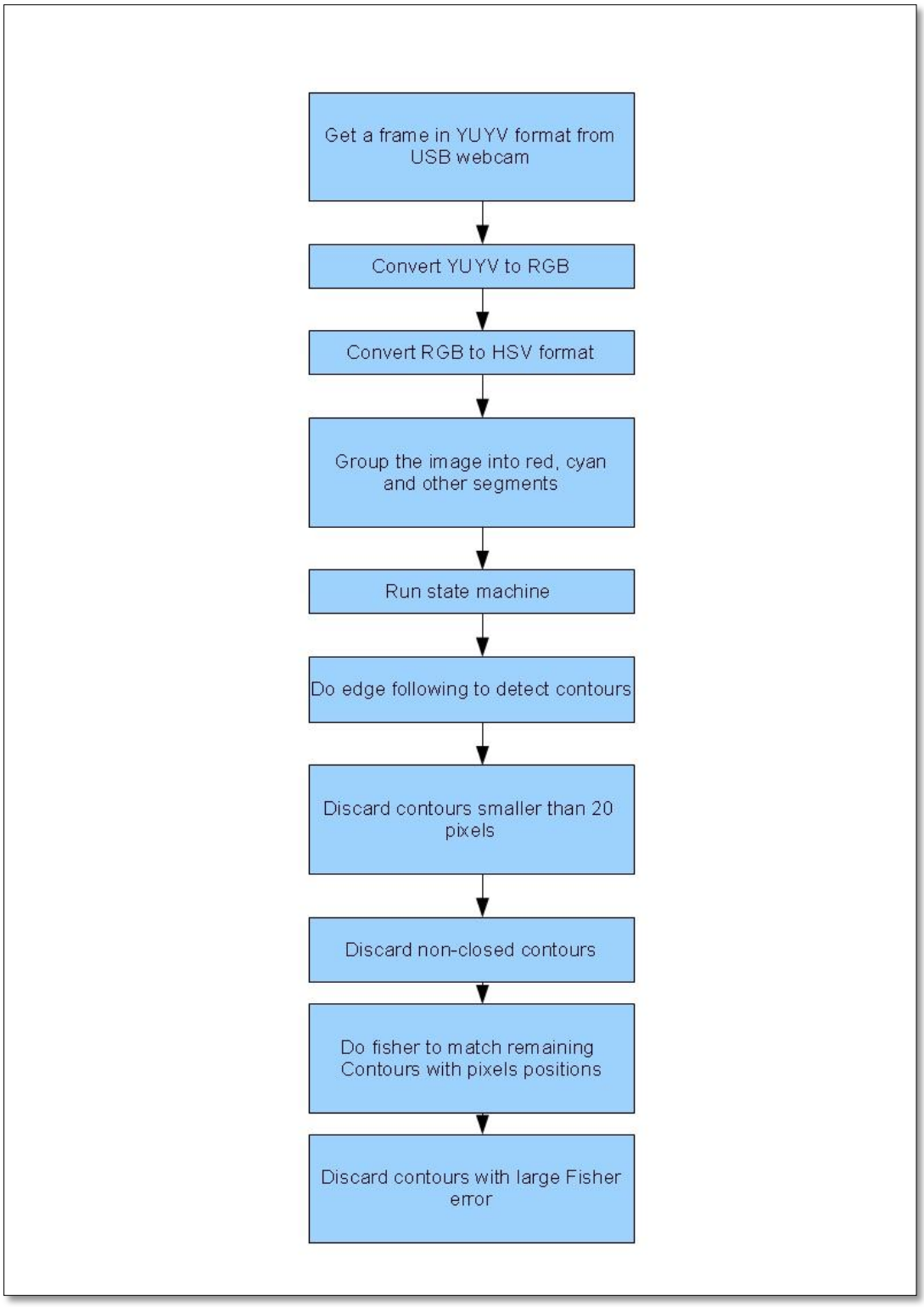


Figure IV.1 – Triangle Detection Architecture

IV.2.3. Convert RGB to HSV format

We do not actually convert the whole frame into HSV format. It is computationally high to convert each frame at every loop. Rather, we create a HSV mapped version of all possible RGB values at the beginning of the program. We use RGB values as indexes to point Hue, Saturation or Value values of the pixels. For HSV color model, see Figure IV.2.

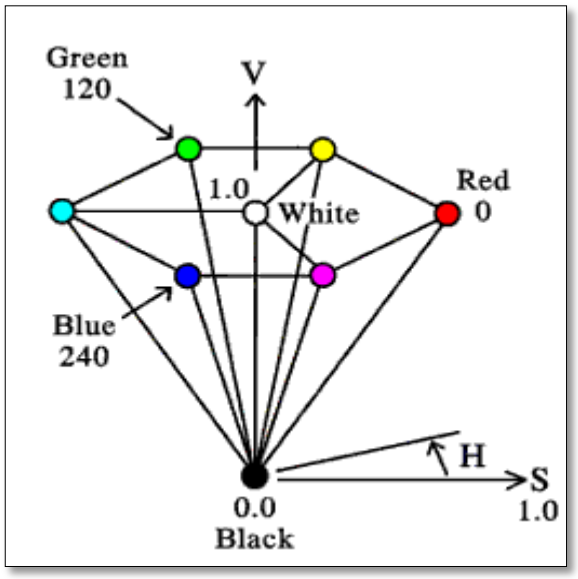


Figure IV.2 – HSV Color Model

IV.2.4. Group the image into Red, Cyan and other segments

In this section, by looking at Hue, Saturation and Value values of each RGB pixels we mark the pixels whether it is red, cyan or non-of-them. We call a do_hsv_check function to return segmented image buffer. We mark the pixels red whose hue values are between -40 and +40 degrees. Also, we mark the pixels cyan whose hue values are between 140 and 220 degrees. In the Figure IV.3, the classification algorithm is depicted. The other pixels are segmented under a third group. See Figure IV.4 for as an example segmented image.

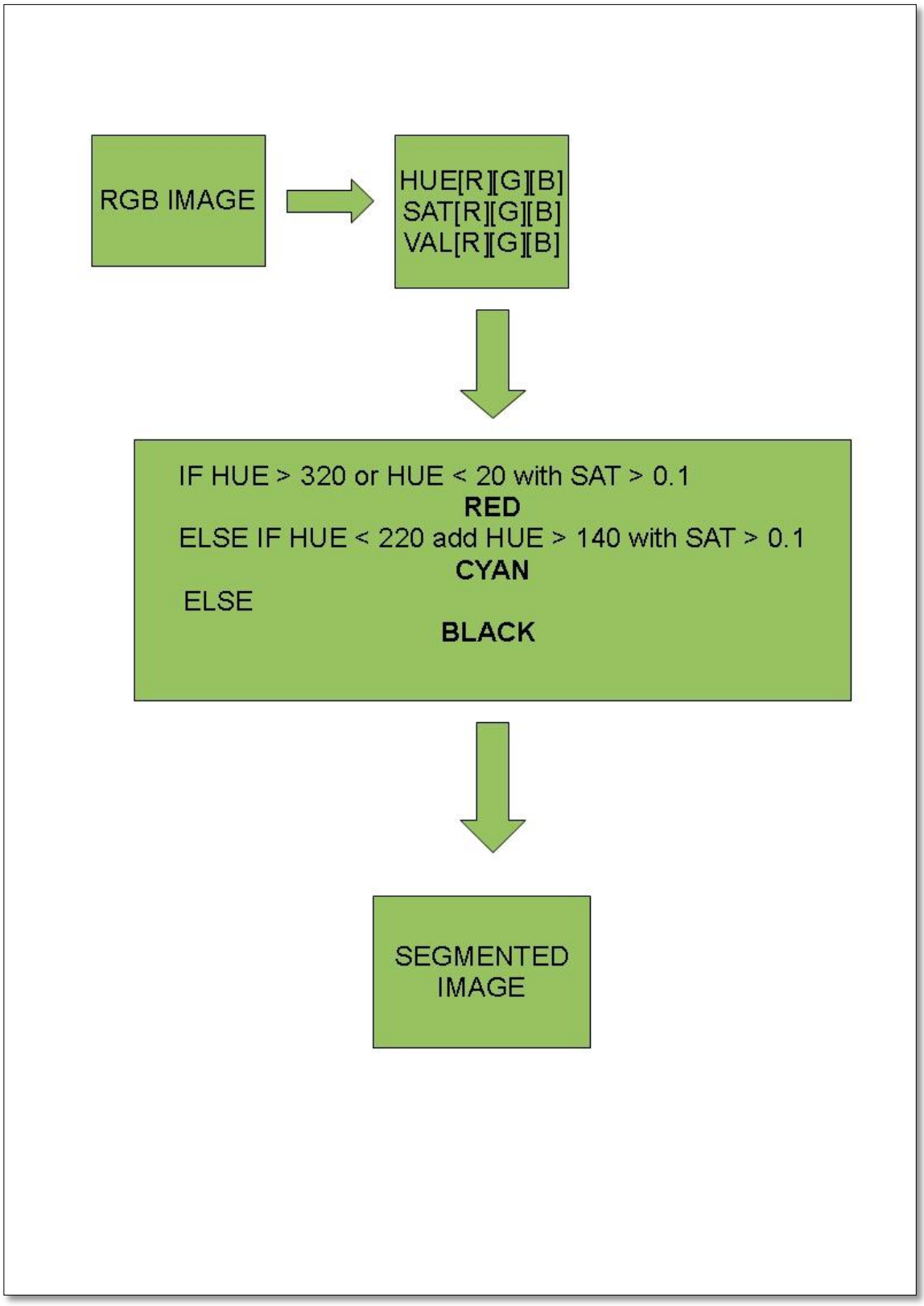


Figure IV.3 Color Segmentation

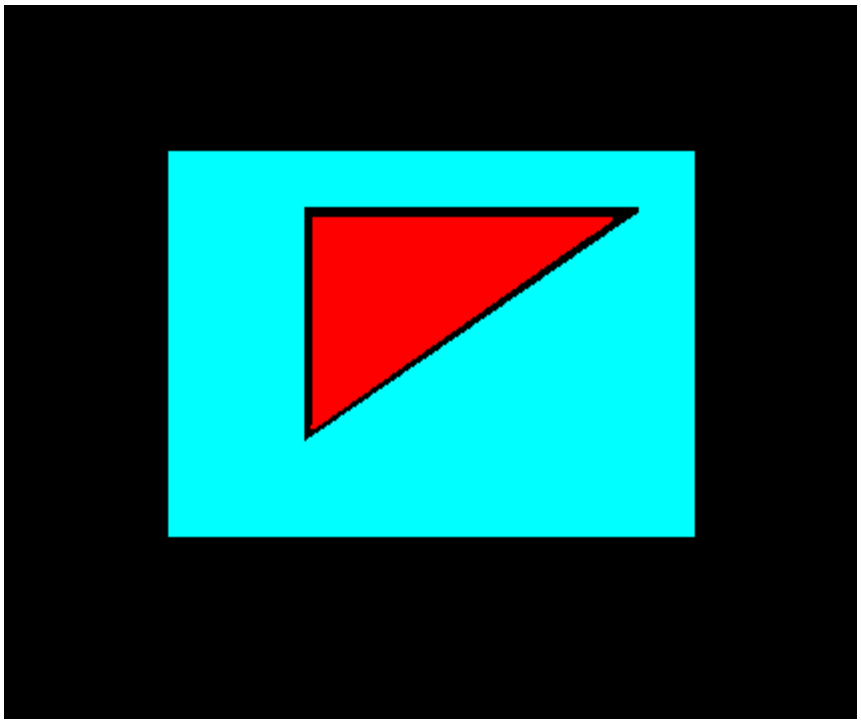


Figure IV.4 Segmented Image

IV.2.5. Runing Color Transition State Machine

Classical approach to edge detection is applying the edge detection filters and mark highly changing regions as edges. However, we do not want to detect all the edges and process each of them. We want to focus on certain regions. We have developed an approach to detect our fiducial to check color transition boundries and upon success mark the boundry as edge. Roughly, we look for cyan-red-cyan trasition points. However, not all time these transitions are detected easily. Motion blur, mis-segmentations result in false pixels between cyan-red or red-cyan transtions. When we check the segmentation frame, we see these regions are black (See Figure IV.4). We call these regions as holes. We implemented the state machine algorithm to detect such holes and overcome the problem. We hold a counter with a upper limit of 30 pixels. If there are maximum 30-pixel hole between red-cyan or cyan-red regions we mark last red and first red pixels as edge respectively. In Figure IV.5, color transition state machine is depicted. And, in Figure IV.6, color transition state machine is run on example segmented image. The pointed pixels in the image are marked as edges as a result of state machine.

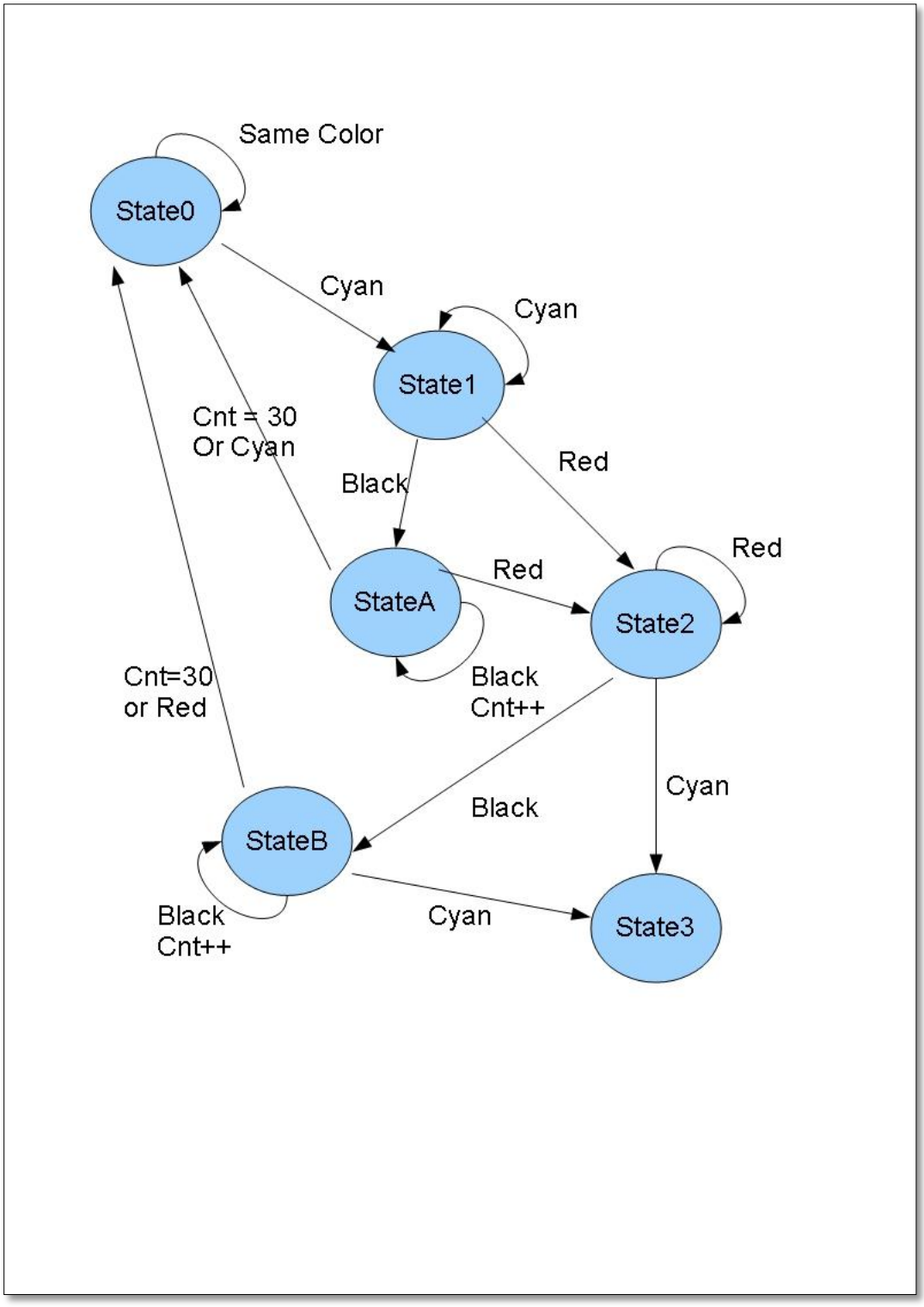


Figure IV.5 Color Trasition State Machine

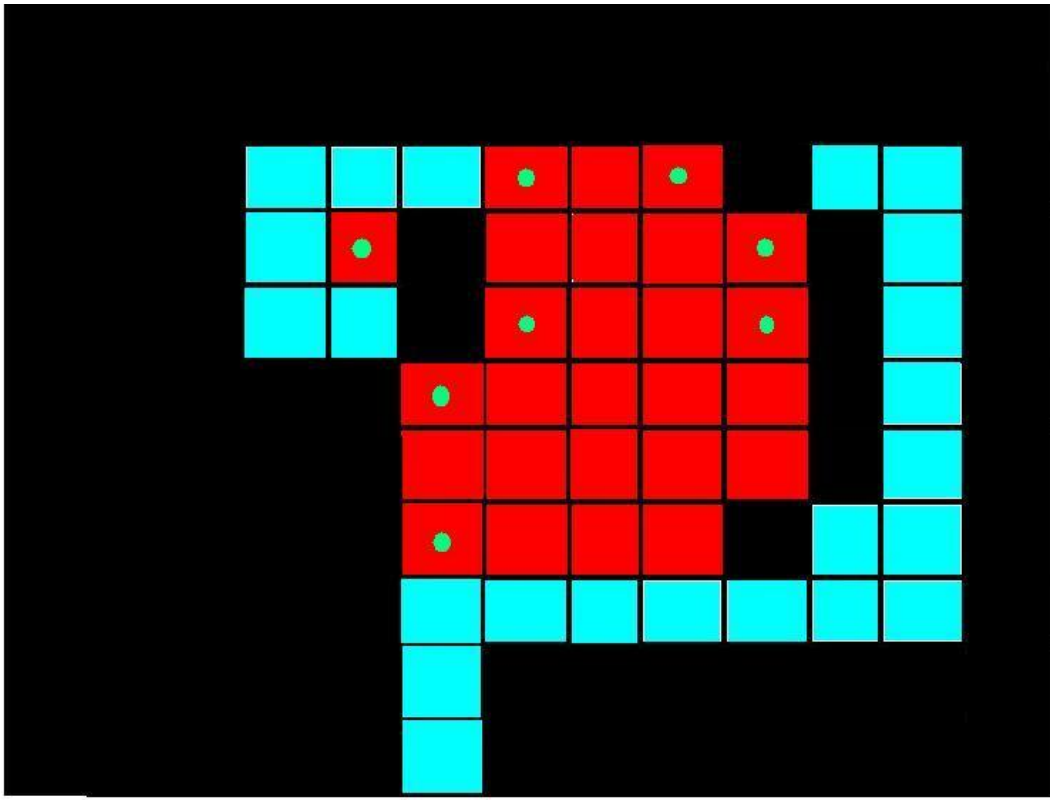


Figure IV.6 - Example Edge Detection

IV.2.6. Do Edge Following To Detect Contours

As as result of state machine step, we have the edge information. In this section, we will group the edge pixels and create linked-list and behave these linked-lists as candidate triangles. We have an edge matrix of the frame. If a pixel is marked as edge, its value in the matrix is set to a value different than 0, else it is set to 0. Thus, to start edge following and contour creation, we search for the first non-zero valued pixels in the matrix. If found, we initate the following process. See Figure IV.7 for linking process. W(x,y) is defined as edge matrix.

Since the linked-list creation is a recursive, neighbour pixels checking can sometimes be ambigous. Such an example, In Figure IV.8, the edge pixels can be located like a ladder shape. Recursion starts from the bottom left pixel. The algorithm then checks for the neighbour edge pixels, It can hit upper neighbour pixel as well as its diagonal neigbour pixel. In Figure IV.8b we see such a false recursion that all diagonal neighbours are added to the linked list before their non-diagonal neighbours. This

approach is errornous in our line-fitting algorithm. The correct recursion order is shown in IV.8a.

```
Start:
Foreach pixel (x,y)
if W(x,y) > 0
        goto LineStart
End
goto Finish
LineStart:
        Create a Linked-list
Testpixel:
        if W(x,y) > 0
                Addpixel to the List
                W(x,y) ← 0
                Goto Testpixel for neighbour pixels
        Endif
  Goto Start
Finish:
```

Figure IV.7- Neigbour Pixel Grouping

To overcome such problem, we have defined criteria for neighbour edge pixel selection. We first select non-diagonal neighbour pixels first, then we look at diagonal pixels. Besides this, our algorithm not only seachs for 1-pixel distant neighbours, but more.

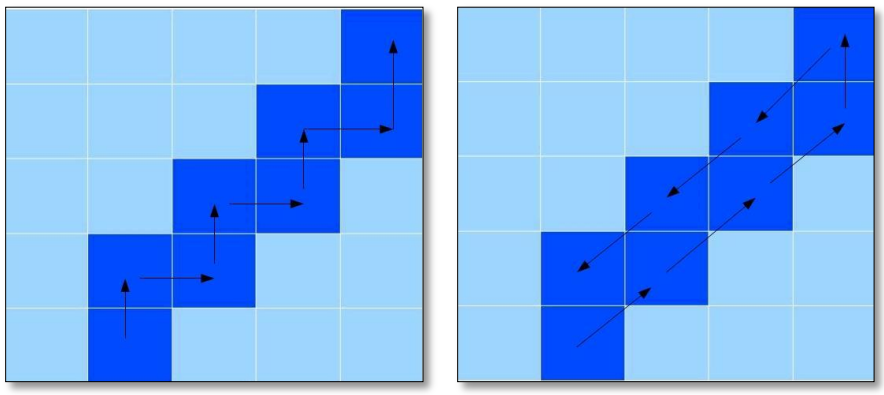


Figure IV.8 a) correct traverse, b) false traverse

It is required to overcome non-continious contours. We are adding pixels to the linked list up to 5 pixel distant. In Figure IV.9 we see neighbour selection. 1-D neighbours are tested first in order: First, the neighbour pixels which are marked as 1, then neighbour pixels which are marked as 2. After 1-D neighbours are tested, 2-D neighbours are tested in order again from the most ortoghonal to the most diagonal ones. In a detailed approach, we look at the square-root distance of the neighbour pixels. The closest ones are selected first.

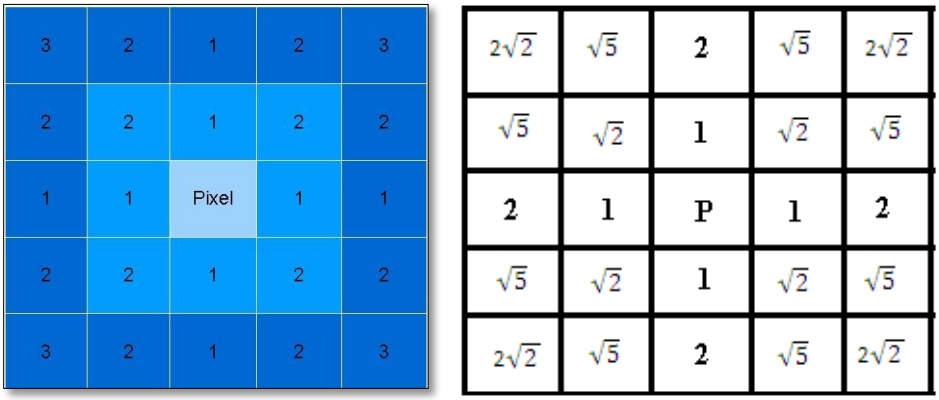


Figure IV.9 Recursion Order of Neighbour Pixels

IV.2.7. Discard contours smaller than 20 pixels

After all neighbour pixels are grouped in contours, we threshold them in number of pixels. If the contour has a smaller number of 20 pixels, we discard it.

IV.2.8. Discard non-closed contours.

After eliminating the contours with a few number of pixels, we check whether the head and tail of the linked list are close eachother. If not, we eliminate them also to avoid confusion of Fisher algorithm, since the straight lines contours also give small errors in line regression.

IV.2.9. Do Linked-List Operations

Before the line fitting algorithm, we have to re-order our linked-list define two corners of the triangle. In Figure IV.10. an example of ambiously create linked-list of countour is depicted. The red numbered edge pixels are added to the list in the printed order, however when adding process hits a discontinouty in the neighbour pixels, the recursion will return back to the starting point, the first pixels. The algorithm will look other neighbour pixels if they are edge pixels. If found, the new pixels will be added to the list in the order pointed in blue. The linked-list creation handles this issue and it reorganizes the pixel-adding process in case of a discontinuty.

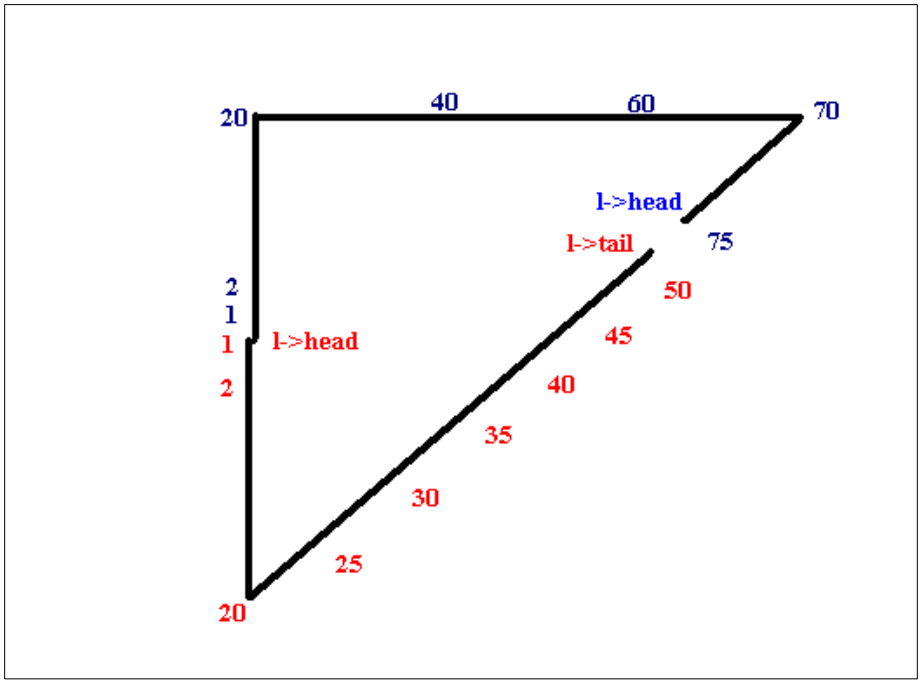


Figure IV.10 – Ambiously created linked-list

After the linked-list is created as desired, we take one more step to align the head of the linked list to one corner of the candidate triangle. From theory, the furthest point from the central point of a triangle is one corner of the triangle. So, we calculate the central point of contour by finding mean (x,y) point of the contour elements.

$$X0 = \frac{\sum x_i}{N}, \quad Y0 = \frac{\sum y_i}{N} \tag{1}$$

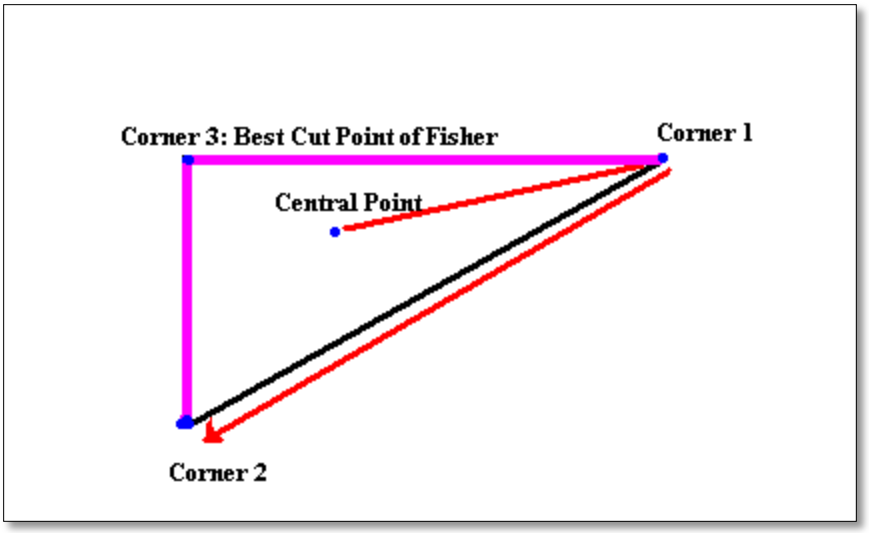


Figure IV.11 - Central and Corner Points

If the contour is a triangle, the most distant pixel on the contour from central point is expected to be one corner of the pixel.

$$argmax_{x,y} C1(x,y) = \sqrt{(x - X0)^2 + (y - Y0)^2} \quad (2)$$

Like in (2), the most distant pixel on the contour from C1 (corner 1) is assumed to be the next corner point of the triangle.

$$argmax_{x,y} C2(x,y) = \sqrt{(x - C1X)^2 + (y - C1Y)^2} \quad (3)$$

In Figure IV.11, we can easily see the central point of the triangle, C1 and C2 corners of the triangle. Though, we can find central, C1 and C2 points for any arbitrary image, they will easily discarded by looking at their line-fitting error since triangle shapes will give low errors.

IV.2.10. Do Fisher to Match Remaining Corner with Pixel Positions.

After finding our C1 corner which is the furthest pixels from the central point of triangle, or arbitrary shape, we set the head (the first pixel of linked list) to this pixel and reorganize the list. However, next pixels of the pixels may locate on different edge of the triangle. In Figure IV.12, it is depicted that linked-list pixels may follow two different paths. The third corner, j, may be located next to C2 (Figure IV.12a) as well we before C2 (Figure IV.12b). In the first case, we will look for the third corner between the pixels *max* and *N-1*. In the second case, between *0* and *max*.

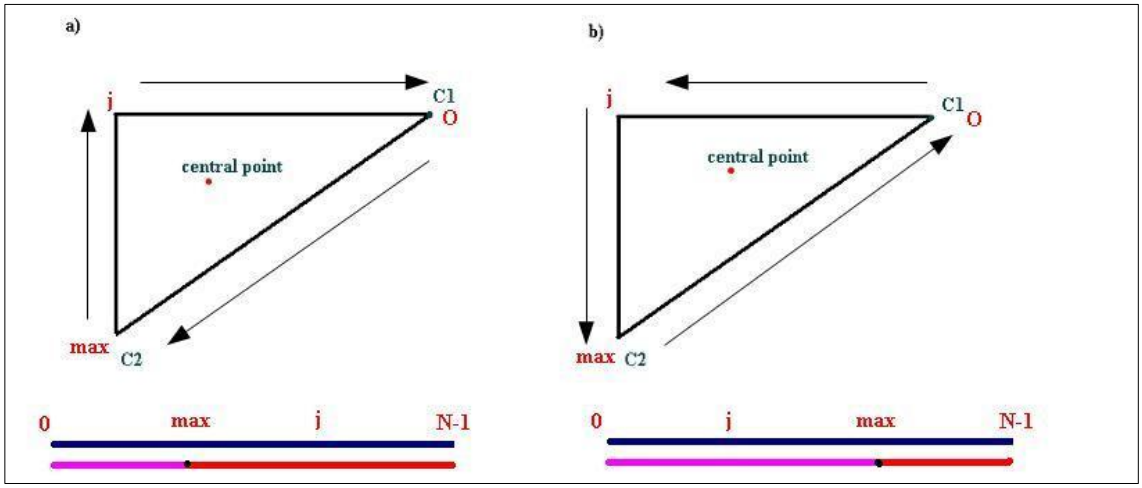


Figure IV.12 – Linked-list Corner Points

To specify at which segment to search for our *j* corner we will look evaluate the lineer regression errors of two segment; *0* - *max* and *max* - *N-1*

$$Err1 = \frac{\sum_{k=0}^{max}(a1x_k + b1y_k + c1)^2}{max}$$

$$Err2 = \frac{\sum_{k=max+1}^{N-1}(a2x_k + b2y_k + c2)^2}{N - max}$$

If *Err1* is smaller than *Err2*, the first edge line between C1-C2 is between the *0*th and *max*th pixels of the linked-list. We will look for the *j*th pixel which is the C3

between *(max+1)*th and *(N-1)*th pixels

If *Err2* is smaller than *Err2*, the first edge line between C1-C2 is between the *(max+1)*th and *(N-1)*th pixels. We will look for *j*th pixel which is the C3 between *0*th and *max*th pixel.

The aim here is to find such *j*th pixel which divides that segment into two segments where two edge lines fit with the minimum error. The error in the first case is:

.

$$Err2 = \frac{\sum_{k=max+1}^{j}(a1x_k + b1y_k + c1)^2}{j - max} + \frac{\sum_{k=j+1}^{N-1}(a2x_k + b2y_k + c2)^2}{N - 1 - j}$$

and in the second case is:

$$Err2 = \frac{\sum_{k=0}^{j}(a1x_k + b1y_k + c1)^2}{j} + \frac{\sum_{k=j+1}^{max}(a2x_k + b2y_k + c2)^2}{max - j}$$

so the total error of three edges of the triangle:

$$Err = Err1 + Err2$$

The a, b, c coefficients are defined two ways. The First method is below.

$$a = -\frac{\sum x_i y_i - \frac{\sum x_i \sum y_i}{N}}{\sum x_i^2 - \frac{(\sum x_i)^2}{N}}$$

$$b = 1$$

$$c = -\frac{a\sum x_i - \sum y_i}{N}$$

The second method is,

$$b = -\frac{\sum x_i y_i - \frac{\sum x_i \sum y_i}{N}}{\sum y_i^2 - \frac{(\sum y_i)^2}{N}}$$

$$a = 1$$

$$c = -\frac{\sum x_i - b \sum y_i}{N}$$

The criteria to select which method is comparing the values $\sum x_i^2 - \frac{(\sum x_i)^2}{N}$, and $\sum y_i^2 - \frac{(\sum y_i)^2}{N}$. If the absolute value of the first is bigger than the second, the first method is selected, otherwise the second.

IV.2.11. Corner Location Validation

For distance calculation of the fiducial, we will create rotation-translation matrix and map position of pixels to the real world coordinate system. We need the corner points of the triangle to be at certain locations. In Figure IV.13, the desired corner

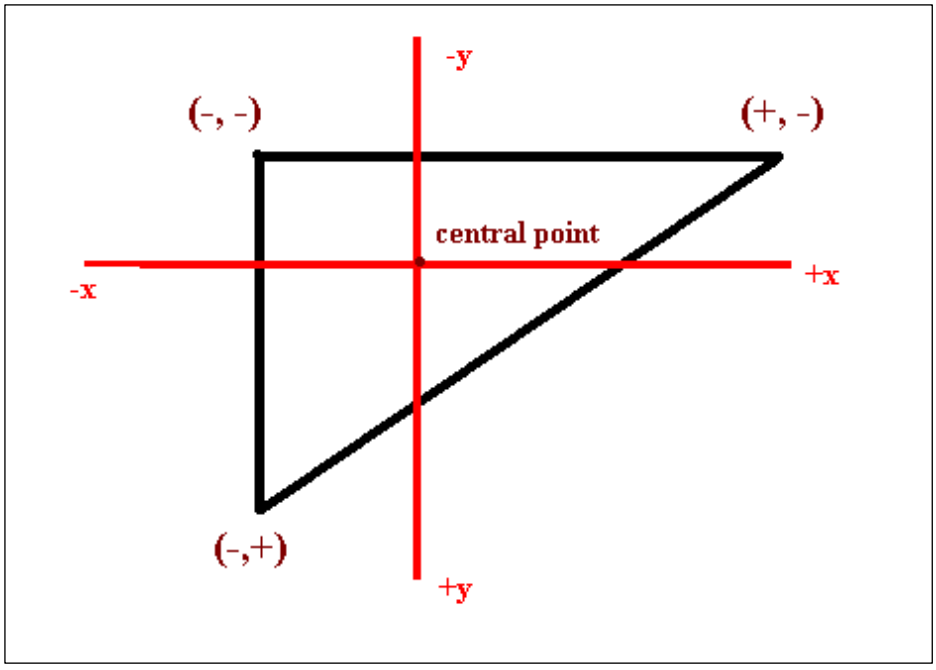


Figure IV.13 – Corner Locations

locations are depicted. In case of one of these criteria fails, the object is elimiated from being triangle.

IV.3. TRIANGLE DETECTION ALGORITHM IMPLEMENTATION

In this section, the detection algorithm will be evalutated in C language form. The main procedure *do_robotics* is called for each frame that is captured from the video device. The function is called with the framebuffer, debugging buffer, height and width parameters. Color segmentation, edge detection, edge thinning, pixel grouping, fisher grouping and triangle desicion processes are initiated in this function.

```
void do_robotics(unsigned char *p, unsigned char *np, int w, int h)
{
      // Some of the definitions are global.
      // The meanings and implementation of
      // the functions will be discussed in
      // C comment blocks.


      hg = h;
      wd = w;

      // do_renk is Color segmentation call. The each pixel in
      // framebuffer is checked depending on its hsv values. Pixels
      // are grouped togeter into three segmens. Red, Cyan or non-of-
      // them

      do_renk(p);


      // Since the code of state machine, it is not included here.
      // this function takes col matrix which is filled with do_renk
      // and searches for certain color transitions. Returns grdw
      // with values set where there edges, cleared values where there
      // is no edge.
      do_state_machine(p, &col,&grdw);


      // After making sure that each candidate edge is represented by
      // a thin, one-pixel-thick contour we start to create connected
      // list of neighbour pixels. Neighbouring criteria may be
      // defined on request, but we chose here it to be 5.

      create_lines(&lines, grdw, wd, hg);

      for(ln = lines.head; ln != NULL; ln = ln->next){
```

```
      // After creating contours, we need to find a starting pixel to
      // begin grouping gradients. We simple choose the furthest pixel
      // from the geometric center. In case of a triangle, it will be
      // a corner point.



                  // We are checking whether head of the contour is
                  // close to the tail of the contour.

                  bason = sqrt((ln->tail->x - ln->head->x)
                              * (ln->tail->x - ln->head->x) +
                             (ln->tail->y - ln->head->y)
                              * (ln->tail->y - ln->head->y));
                  if(bason < 6 && (ln->errn < 0.5)
                        && (nmax == -1 || nmax < ln->nofpixels)){
                        find_startings(ln);
                        fisher_grouping(ln, wd, hg);
                        lmax = ln;
                        nmax = ln->nofpixels;
                  }
                  bason = 0;
      }
      if(lmax != NULL){
        triangle_points(output_ptr, lmax, wd, hg);
        if(lmax->is_triangle){
            find_distance(lmax, wd, hg);
            mark_lines(output_ptr,lmax, wd, hg);
        }
        if(is_histogram)
                  histogram_plot(lmax, output_ptr);
      }

      clean_all(&lines);
}
```

Do_hsv_check function takes r,g,b values as parameters and specifies if r,g,b values form a red-color pixel. By doing so, r,g,b values are converted into HSV format. Function is dynamic to return requested hue value. If the pixels is accepted to be around hue value, the function returns true.

```
int do_hsv_check(int r, int g, int b,
             float huu, float saa, float val, // 0, 1, 100
               float ddh, float dds, float ddv)
{
      float hup, hdown, ds, dv;
      float hcheck = 0;
```

```
	unsigned char print_buf[1024];


	hup = (huu + ddh);  // 10 + 20  = 330
	if(hup > 360)
		hup -= 360;

	hdown = (huu - ddh);

	if(hdown < 0)
		hdown = 360 + hdown;


	if(hdown > hup){
		if(hue[r][g][b] >= hdown || hue[r][g][b] <= hup){
			hcheck = 1;
		} else {
			hcheck = 0;
		}
	} else {
		if(hue[r][g][b]  <= hup &&  hue[r][g][b] > hdown){
			hcheck = 1;
		} else {
			hcheck = 0;
		}
	}

	if(hcheck == 1 && sat[r][g][b] > dds && value[r][g][b] > ddv)
		return 1;

	return 0;


}
```

In the following section, we will be explaining contour extraction from the gradient values. Before this process, the candidate edges are thinned depending on the criteria above.

Colors are for display and debug purposes. Each connected contour is represented by a differend color from the pool below.

```
// colors of contours on the display
int colors[8][3] = {{0x00, 0x00, 0x00},    // black
                {0xff, 0x00, 0x00},  // red
                {0xff, 0xff, 0x00},  // orange
                {0xff, 0x00, 0xff},  // magenta?
                {0x00, 0xff, 0x00},  // green
```

```
            {0x00, 0xff, 0xff},  // yellow
            {0x00, 0x00, 0xff},  // blue
            {0xff, 0xff, 0xff}}; // white
```

To group pixels, we need to find a starting point. As we stated above starting poing is selected to be the furhest pixel.

```
void find_startings(struct line *l)
{
      struct pixel *p;
      unsigned int ox, oy;
      int dx, dy;

            l->ox = l->oy = 0;

            for(p = l->head; p != NULL; p = p->next){
                  l->ox += p->x;
                  l->oy += p->y;
            }
            l->ox /= l->nofpixels;
            l->oy /= l->nofpixels;
            l->Rmax = -1;
            for(p = l->head; p != NULL; p = p->next){
                  dx = p->x - l->ox;
                  dy = p->y - l->oy;
                  p->R = sqrt((dx * dx) + (dy * dy));
                  if(l->Rmax == -1 || p->R > l->Rmax){
                        l->Rmax = p->R;
                        l->rhead = p;
                  }
            }

            int maxc1;
            int tmax;

            maxc1 = -1;

            int s;

            for(p = l->head; p != NULL; p = p->next){
                  dx = p->x - (l->rhead)->x;
                  dy = p->y - (l->rhead)->y;
                  tmax = (dx * dx) + (dy * dy);
                  if(maxc1 == -1 || maxc1 < tmax){
                        l->c1p = p;
                        l->c1x = p->x;
                        l->c1y = p->y;
                        maxc1 = tmax;

                  }
            }
            int stop = 0;
            p = l->rhead;
            while(stop == 0){
```

```
                if(p->next == NULL){
                        p->next = l->head;
                        l->head->prev = p;
                        p = l->head;
                } else if(p->next == l->rhead){
                        p->next = NULL;
                        l->head = l->rhead;
                        l->head->prev = NULL;
                        l->tail = p;
                        stop = 1;
                } else {
                        p = p->next;
                }
        }
        list2array(l);


}
```

Create_lines simply groups neighbour pixels together to form contours. Each contour is independent set of pixels. After create_lines is called, the function search for a non-zero weighted gradient value. Upon success, a new linked list is initated. Plus, the pixel is added to the newly created list. Next, the neighbour pixels which are one pixel distant are checked whether they have valid gradient values . If so, the neighbour pixel is added to the list and it is chosen to be the current pixel,

```
void create_lines(struct lines *ln, int **grdw, int wd, int hg)
{
      struct spix sp;
      struct line *l;
      int x,y,s;
      int i;
      while(search_pixel(grdw, wd, hg, &sp) == 1){
            l = (struct line *)malloc(sizeof(struct line));
            l->head =  NULL;
            l->tail = NULL;
            l->next = NULL;
            l->nofpixels = 0;
            l->mode = 0;
            l->depth = 0;
            l->p1 = l->p2 = l->p3 = NULL;
            l->is_triangle = 0;
            l->lp1 = l->lp2 = l->lp3 = 0;
            l->p = (struct pixel *)malloc(12);
            l->p[0] = (struct pixel *)malloc(sizeof(struct pixel));
            l->p[1] = (struct pixel *)malloc(sizeof(struct pixel));
            l->p[2] = (struct pixel *)malloc(sizeof(struct pixel));
            for(i = 0; i < 360; i++){
                  l->angle[i] = 0;
                  l->nofan[i] = 0;
            }
```

```
            memcpy(&l->color, &colors[ln->noflines%8], 3);
            if(ln->noflines == 0){
                  ln->tail = l;
                  ln->head = l;
                  ln->noflines = 1;
            } else {
                  ln->tail->next = l;
                  l->prev = ln->tail;
                  ln->tail = l;
                  ln->noflines++;
            }
            add_pixel(l, NULL, grdw, wd, hg, sp.x, sp.y);
            if(l->nofpixels <  NMINCON)
                  remove_line(ln, l);
      }
}
```

Add_pixel adds the pixels to the linked list and clears it in the gradient weight matrix not to evaluate it again later. This means, one pixel is used once throughout the frame.

```
void add_pixel(struct line *l, struct pixel *pp, int **grdw,
            int wd, int hg, int x, int y)
{
      struct pixel *p;
      int depth;
      int srt;

       depth = 0;

       if(x < wd && y < hg &&
         x > -1 && y > -1 &&
         grdw[x][y] > 0){
                  p = (struct pixel *)malloc(sizeof(struct pixel));
                  bzero(p, sizeof(struct pixel));
                  p->x = x;
                  p->y = y;

                  grdw[x][y] = 0;
                  p->next = NULL;
                  p->prev = NULL;
                  p->neigh = 0;
                  if(l->mode == 0){
                        if(l->nofpixels == 0){
                              l->head = p;
                              l->tail = p;
                              l->nofpixels = 1;
                        } else if(l->tail == pp){
                              l->tail->next = p;
                              p->prev = l->tail;
                              l->tail = p;
                              l->nofpixels++;
                        } else {
                              p->next = pp->next;
                              pp->next->prev = p;
```

```
                              p->prev = pp;
                              pp->next = p;
                              l->nofpixels++;
                        }
                  } else if(l->mode == 1){
                        p->next = l->tail;
                        p->prev = l->tail->prev;
                        l->tail->prev->next = p;
                        l->tail->prev = p;
                        l->nofpixels++;
                  }

      } else
            return;


      while(depth < global_depth){
            depth++;
            add_DN_pixels(l, p, grdw, wd, hg, depth);
            if(p == l->head && l->nofpixels > 1)
                  l->mode = 1;
      }

}
```

Add_DN_pixels calls for the possible neighbour pixels starting from 1 to N pixel further away. Next neighbour selection is described in section IV.2.6.

```
void add_DN_pixels(struct line *l, struct pixel *p,
            int **grdw, int wd, int hg, int N)
{
      int i, j;
      int t;
      int x1, y1, min;

      for(t = 0; t <= N; t++){
            for(i = -N; i <= N; i++){
                  for(j = -N; j <= N; j++){
                     if(i == N || j == N
                       || i == -N || j == -N){
                        x1 = i;
                        y1 = j;
                        if(x1 < 0)
                              x1 = -x1;
                        if(y1 < 0)
                              y1 = -y1;
                        if(x1 < y1)
                              min = x1;
                        else
                              min = y1;
                        if(t == min)
                              add_pixel(l, p, grdw,
                                wd, hg,
                                p->x + i,
                                p->y + j);
```

```
                        }
                    }
                }
        }

}
```

Search_pixel checks the global gradient matrix to find remaning non-zero gradient valued pixels.

```
int search_pixel(int **grdw, int wd, int hg, struct spix *spx)
{
        int x,y,s;

        for(x = 0; x < wd; x++){
                for(y = 0; y < hg; y++){
                        if(grdw[x][y] != 0){
                                spx->x = x;
                                spx->y = y;
                                return 1;
                        }
                }
        }
        return 0;
}
```

Remove_line simply removes a line from lines liked list. An example case for this call is checking the number of pixels in the line and if the number of pixels is below a threshold value, it is removed from further evaluation.

```
void remove_line(struct lines *ln, struct line *l)
{
        struct pixel *p;
        struct pixel *n;

        if(ln->head == l){
                ln->noflines = 0;
                ln->head = NULL;
                ln->tail = NULL;
        } else {
                l->prev->next = NULL;
                ln->tail = l->prev;
                ln->noflines--;
        }

        p = l->head;
        while(p){
                n = p->next;
                free(p);
                p = n;
        }
```

```
	free(l);

}
```

Fisher_grouping is the essential function which groups gradient angles in the linked list into three subgroups where each group is “likely” to eachother. The overal problem here is minimizing the total variance thus error and finding 2 cut points which realizes this aim.

```
void fisher_grouping(struct line *l, int wd, int hg)
{

	int i,j,k,x,y,z, s;
	struct pixel *px;
	double mean1, mean2, mean3;
	double q1, q2, q3;
	double qt = -1;
	double xx;
	time_t saniye;
	unsigned short millitm;
	unsigned char buf[1024];
	int base;
	float Alocal, Blocal, Clocal;
	float A1local, B1local, C1local;
	float alocal, blocal, clocal;
	int man;
	int N;
	float vstatic;

	int inx = 0;

	l->err = -1;
	qt = -1;
	xx = 0;

	double v1, v2,v3, v4;

	base = 0;
	Sx = (double *)malloc(sizeof(double)
				* l->nofpixels);
	Sy = (double *)malloc(sizeof(double)
				* l->nofpixels);
	Sx2 = (double *)malloc(sizeof(double)
					* l->nofpixels);
	Sxy = (double *)malloc(sizeof(double)
				* l->nofpixels);
	Sy2 = (double *)malloc(sizeof(double)
					* l->nofpixels);
```

```
for(i = 0; i < l->nofpixels; i++){
      Sx[i] = 0;
      Sy[i] = 0;
      Sx2[i] = 0;
      Sxy[i] = 0;
      Sy2[i] = 0;

}

Sx[0] = l->arr[0]->x;
Sy[0] = l->arr[0]->y;
Sx2[0] = Sx[0] * Sx[0];
Sxy[0] = Sx[0] * Sy[0];
Sy2[0] = Sy[0] * Sy[0];
for(i = 1; i < l->nofpixels; i++){
      Sx[i] = Sx[i-1] + l->arr[i]->x;
      Sy[i] = Sy[i-1] + l->arr[i]->y;
      Sx2[i] = Sx2[i-1] + (l->arr[i]->x *
                        l->arr[i]->x);
      Sy2[i] = Sy2[i-1] + (l->arr[i]->y *
                        l->arr[i]->y);
      Sxy[i] = Sxy[i-1] + (l->arr[i]->x *
                        l->arr[i]->y);
}


for(i = 0; i < l->nofpixels; i++)
      if(l->arr[i] == l->c1p){
            man = i;
            break;
      }

N = l->nofpixels;

v1 = verror(l, 0, man);
v2 = verror(l, man, l->nofpixels - 1);

qt = -1;
float A1, B1, C1;
float A1S, B1S, C1S;
float A2, B2, C2;
float A2S, B2S, C2S;
int i_save, j_save, k_save;

A1 = B1 = C1 = A2 = B2 = C2 = 0;

if(v1 > v2){
      vstatic = v2;
      for(i = 1; i < man-1; i++){
            v1 = find_line(l, 0, i, &A1, &B1, &C1);
            v2 = find_line(l, i, man, &A2, &B2, &C2);
            if(qt == -1 || (v1 + v2) < qt){
                  qt = v1 + v2;
                  l->p[0] = l->arr[0];
                  l->p[1] = l->arr[man];
                  l->p[2] = l->arr[i];
```

```
                        l->a1 = A1;
                        l->b1 = B1;
                        l->c1 = C1;
                        l->a2 = A2;
                        l->b2 = B2;
                        l->c2 = C2;
                        l->l1s = man;
                        l->l1e = N-1;
                        l->l2s = 0;
                        l->l2e = i;
                        l->l3s = i;
                        l->l3e = man;
                  }

            }
            vstatic += qt;
            l->a0 = A(man, N-1);
            l->b0 = B(man, N-1);

      } else {
            vstatic = v1;
            for(i = man + 1; i < N-1; i++){
                  v1 = find_line(l, man, i, &A1, &B1, &C1);
                  v2 = find_line(l, i, N-1, &A2, &B2, &C2);
                  if(qt == -1 || (v1 + v2) < qt){
                        qt = v1 + v2;
                        l->p[0] = l->arr[0];
                        l->p[1] = l->arr[man];
                        l->p[2] = l->arr[i];
                        l->a1 = A1;
                        l->b1 = B1;
                        l->c1 = C1;
                        l->a2 = A2;
                        l->b2 = B2;
                        l->c2 = C2;
                        l->l1s = 0;
                        l->l1e = man;
                        l->l2s = man;
                        l->l2e = i;
                        l->l3s = i;
                        l->l3e = N-1;
                  }
            }
            vstatic += qt;
            l->a0 = A(0, man);
            l->b0 = B(0, man);

      }

      l->err = vstatic;
      l->errn = (float)l->err/(float)(l->nofpixels);

      free(Sx);
      free(Sy);
      free(Sxy);
      free(Sx2);
      free(Sy2);
```

List2array is taking linked list and make an array of them. It is useful for directly adddressing the required pixels in the contour, otherwise, we should scan the linked list and extract the structure each time.

```
void list2array(struct line *l){
      struct pixel *p;
      int i;

      l->arr = (struct pixel *)malloc(sizeof(struct pixel) *
                              l->nofpixels);

      i = 0;
      p = l->head;
      while(i < l->nofpixels){
            l->arr[i] = p;
            p = p->next;
            if(p == NULL)
                  p = l->head;
            i++;
      }
}
```

Triangle_points is where 3 points forms a triangle or not.

```
void triangle_points(unsigned char *p, struct lines *ln, int wd, int
hg)
{
      struct line *l;
      struct pixel *px;
      float err;
      int i,j, t;
      int s;

      for(l = ln->head; l != NULL; l = l->next){
            err = l->err/((float)l->nofpixels);
            if(l->nofpixels > NMINCON  && err < 3000){
                  for(t = 0; t < 3; t++)
                        for(i = l->p[t].x - 2;
                            i < l->p[t].x + 2; i++)
                              for(j = l->p[t].y - 2;
                                  j < l->p[t].y + 2; j++){
                                    if(i > -1 && j > -1 &&
                                       i < wd && j < hg){
                                          s = (i + (j * wd)) * 3;
                                          p[s] = 0xff;
                                          p[s+1] =  0xff;
                                          p[s+2] = 0x00;
                                    }
                              }
            }

      }
}
```

CHAPTER V

PROJECTION AND DISTANCE CALCULATION

V.1. INTRODUCTION

After we find the x, y values of the corner points of the triangle, we have to convert these points to 3D space to address the real location of the triangle related to the camera, thus to the vehicle. Here, we need two parameters: one is distance of the triangle and the second is the angle between the center point of the triangle and the z-axis as seen in Figure V.1.

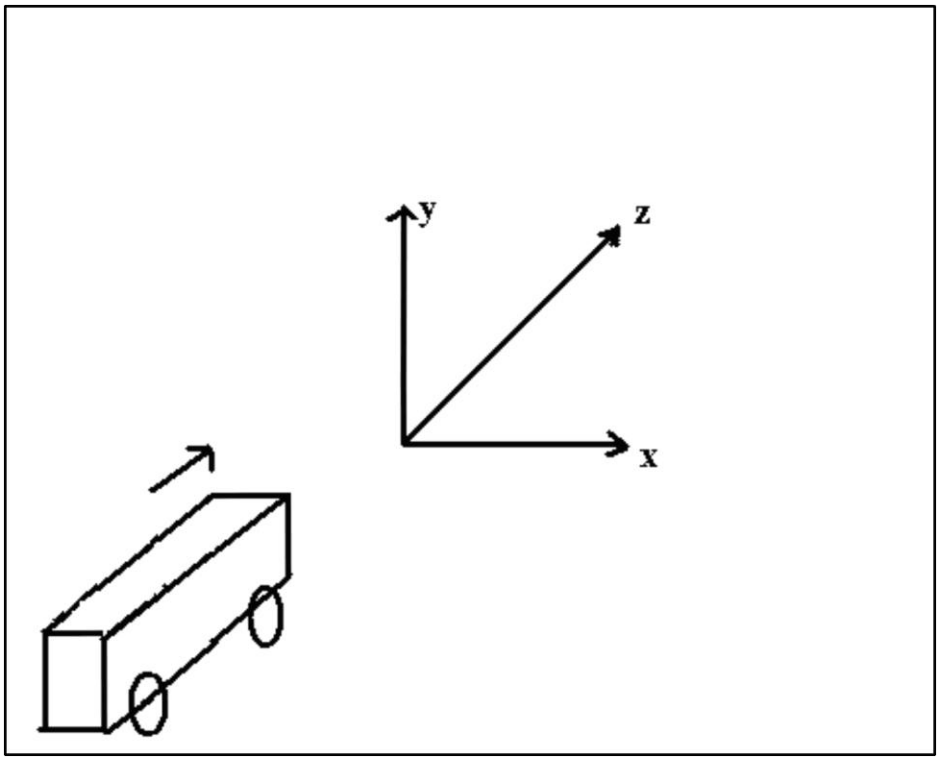


Figure V.1 – Vehicle and 3D axises

In camera system, objects are projected depending on the focal length of the lense used. We have estimated a K scalar including focal length f using several examples depending on the formula below:

$$\frac{d}{D} = \frac{p}{K}$$

Where d is real size of the object, D is the distance of object from the camera, p is size of object in pixels and K is a scalar we want to estimate, which we use, after calibration step for calculating position of each objects.

To avoid from misunderstanding, from now on we will use the notation (u,v) location of each point in image plane, where u is position on x axis and v is position of the point on y axis. Thus the equation becomes:

$$u = K\frac{X}{Z}, \qquad v = K\frac{Y}{Z}$$

Where X, Y, Z are positions in real world coordinate system whereas u and v are positions in 2-d image plane. As we have stated that X, Y, Z are real world coordinates of object, these values are translated and rotated versions of original object.

Since calculation of 3-axis rotation is costly, we accept our object to rotate only around Y-axis. Thus, the rotation and translation matrix becomes as below:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & x_d \\ 0 & 1 & 0 & y_d \\ 0 & 0 & 1 & z_d \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\beta & 0 & -sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ sin\beta & 0 & cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

where x,y,z values are real coordinates of a corner point which we define and never chances. The coordinates of the corner points of our fiducial triangles are (0, 0, 0), (0,-208, 0), (297, 0, 0) like in Figure V.2. *K* is our camera scalar which is 521.5.

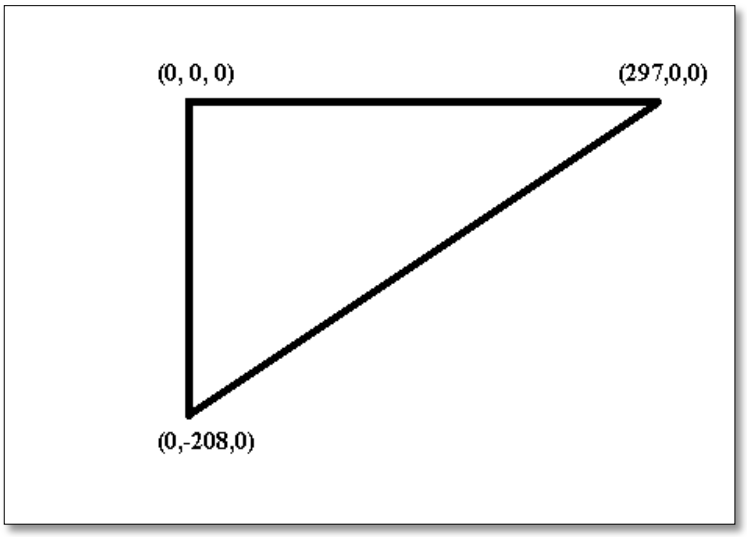


Figure V.2 – Real Coordinate Points of the Triangle

The relation between the points in image plane and the real world is as below. x_1', y_1', z_1' values are translated-and-rotated versions of the first corner point in 3-D coordinate.

$$u_1 = K \cdot \frac{x_1'}{z_1'}$$

$$v_1 = K \cdot \frac{y_1'}{z_1'}$$

If we combine translation-and-rotation matrix and 2D-to-3D mapping equations of all corner points, we get the matrix equation below where $x_{d,}, y_d, z_d$ values are unknown and aimed to solved.

$$\begin{bmatrix} K & 0 & -u_1 \\ 0 & K & -v_1 \\ K & 0 & -u_2 \\ 0 & K & -v_2 \\ K & 0 & -u_3 \\ 0 & K & -v_3 \end{bmatrix} \begin{bmatrix} x_d \\ y_d \\ z_d \end{bmatrix} = \begin{bmatrix} u_1 sin\beta x_1' - K cos\beta x_1' \\ v_1 sin\beta - K y_1' \\ u_2 sin\beta x_2' - K cos\beta x_2' \\ v_2 sin\beta - K y_2' \\ u_3 sin\beta x_3' - K cos\beta x_3' \\ v_2 sin\beta - K y_3' \end{bmatrix}$$

Takes the form of,

$$Ax = B$$

the solution of *x* is,

$$x = (A^T A)A^T B$$

After solving the equation below, we calculate the fit error as below:

$$Err = \sum_{i=1}^{3} (u_i - K\frac{cos\beta x_i' + x_d}{sin\beta x_i' + z_d})^2 + (v_i - K\frac{y_i' + y_d}{sin\beta x_i' + z_d})^2$$

We try to minimize the error with the best suitable β angle. Our rangle is (-45, 45). After finding x_d, y_d, z_d we calculate the distance of the object as $\sqrt{x_d^2 + y_d^2 + z_d^2}$ and the orientation of the object as $\cos^{-1}\frac{x_d}{\sqrt{x_d^2 + z_d^2}}$.

We instruct the steering and driving system with the parameters of distance and direction. We have mapped angle values of -20 to 20 rangle to serial byte 20 – 120. The speed is determined the criteria below:

$$Speed = \begin{cases} distance < 2m & 129, \\ 2m < distance < 5m & 125\frac{distance - 2m}{5m} + 129, \\ distance > 5m & 254 \end{cases}$$

CHAPTER VI

DRIVING SUBSYSTEM

VI.1. INTRODUCTION

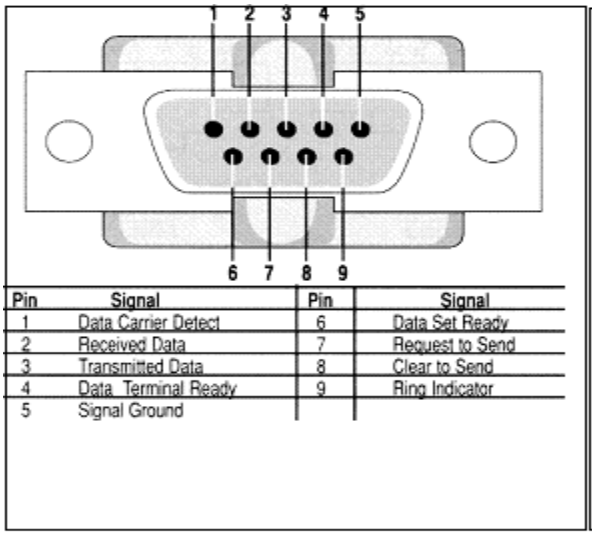
In this section, we will be discussing about the subsystem which drives the car in both changing the speed and the direction. This part of the vehicle is basically composed of electronic circuitry and DC electric motors. Processing unit sends speed and direction information via serial communication to the electronic circutry which includes one microchip pic controller, one H-bridge and electronic components. The serial data is recieved by pic controller and checked whether the recieved data is speed or direction. If it is direction, the controller generates a signal to directly attached H-bridge chip which runs the DC motor that is on the front wheel.

If the data is speed, the controller generates PWM signal to switch on/off the power transistors to drive the rear motors . We will detail the how the circuitry works in the following sections.

VI.2. SERIAL COMMUNICATION

Communication between the processing unit and the driving logic is accomplished over serial (DB-9S) communication protocol. We are using one way transfer from processing unit to pic based control circuit, via which the speed and the direction parameters are sent from computer to logic. RS232 DB-9S has 9 pins which are organized like in Figure VI.1. We have used pins that are numbered 3 (Transmitted Data), and 5 (Ground) for transmitting data from computer to the control logic. The serial communication parameters are 9600 BAUD, parity on, stop bits. These parameters are set both in the driving algorithm in the processing unit and receiving algorithm in Microchip microcontroller to provide data transfer. On the circuit side, a

resistor between the pin 4 and 6 is placed to activate RS232 communication. Without this resistor, the computer will not detect any serial device.



Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

Figure VI.1: a) RS232 Pin Layout and b) RS232 Cable and Sockets

Serial communication is the process of sending data one bit at a time over a communication channel or computer bus [11]. When the serial line is idle, which means that there is no data being transmitted, the value of the line is set to 1. For a 8-bit data to be sent over the line, the signal on the line is cleared, than the data is sent in 8 bits, consequently, the stop bit is set, lastly, the line is set to 1, idle. Protocol implementation is out of scope of this thesis, thus simply is left to operating system itself.

VI.3. DRIVING ALGORITHM

The driving algorithm recieves data over serial communication line from computer and evaluating the recieved data, it decides whether to drive the front wheels or rear wheels. The value of the recieved data is from 0 to 255. Roughly, the high most bit determines which wheels will be driven. If the first bit is 0, than the last 7 bits are used as angle information of the front wheel, and vice versa, if the first bit is 1, the last

7 bits will be used as the speed of the vehicle. In the first case, 7 bits, thus 128 values are mapped to the angle values from -64 to 63, accepting 0 is moving straight ahead. In the second case, speed levels are between 0 (stop) and 128 (maximum speed ahead).

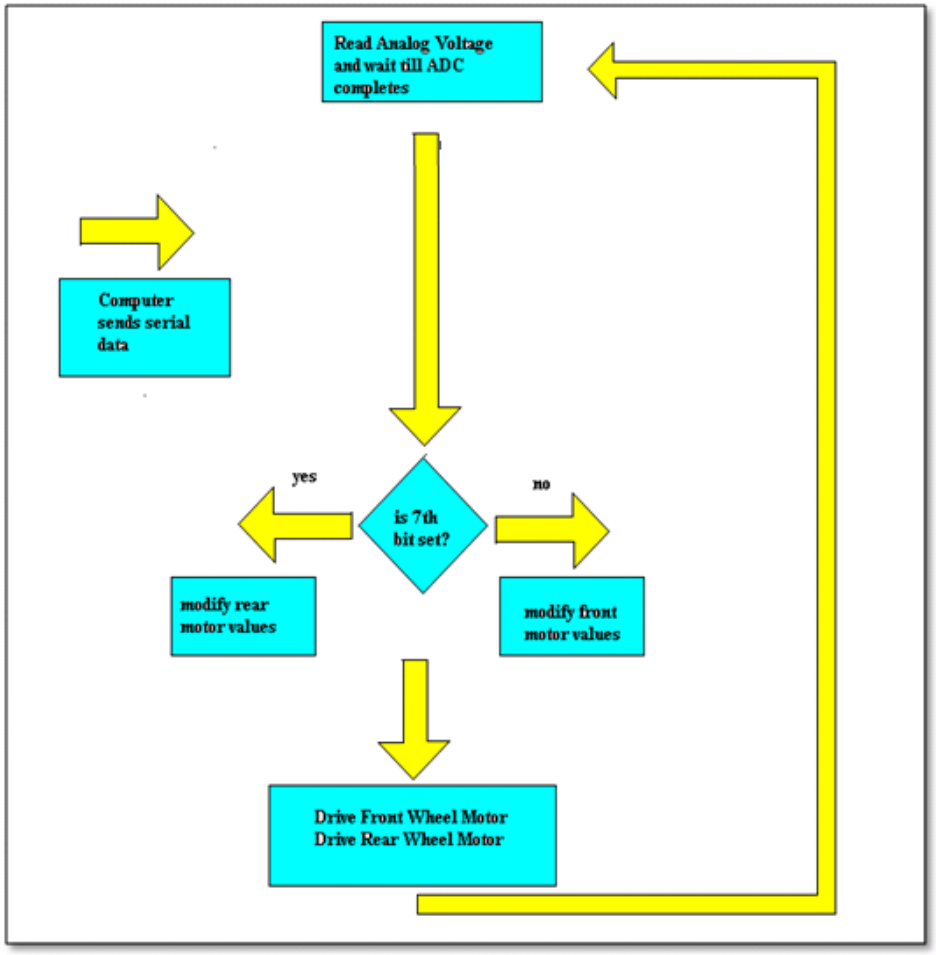


Figure VI.2 – Motor Driving Architecture

VI.4. DRIVING CIRCUIT

In Figure VI.3, the driving circuit is located. Steering motor is physically tied to the resistor to provide position feedback to the pic. The Processing Unit sends steering and speed data to the PIC, then the PIC generates front control to the H-bridge (BA6209) . Also PIC creates a PWM signal to the rear motor control circuit.

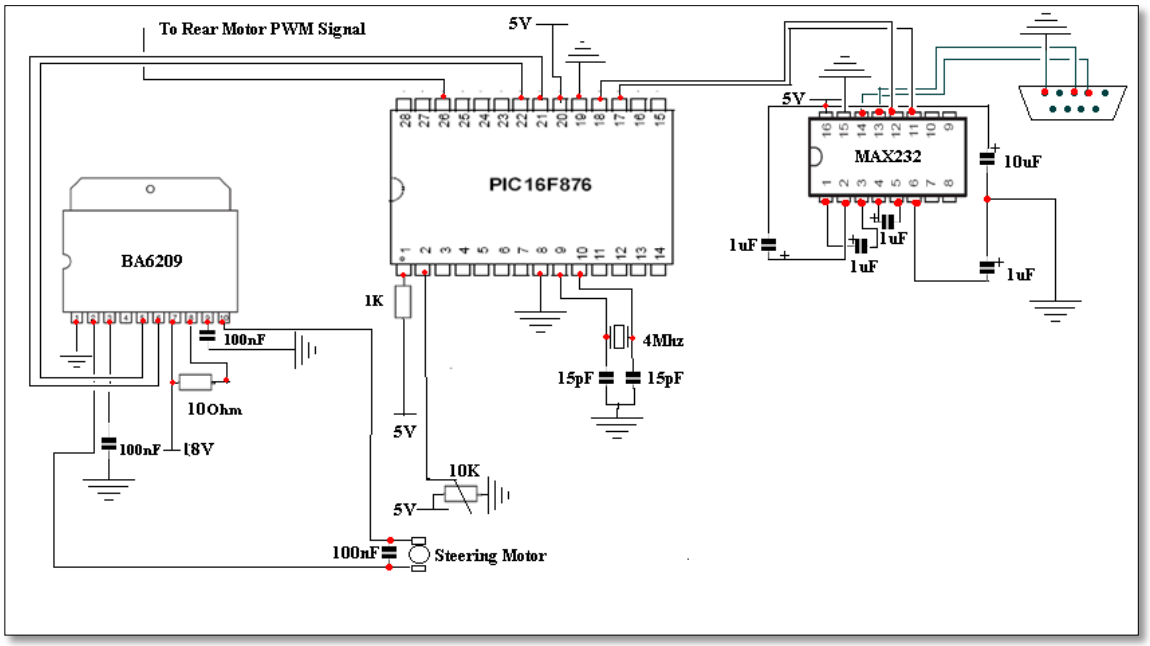


Figure IV.3 – Motor Control Circuit 1

In Figure VI.4, Steering motor control signals are exampled. PB0 is located at 21st pin of PIC16F876 and PB1 is located at 22nd pin. When PB0 is set and PB1 is cleared, the H-Bridge drives front motor to the left, the potentiometer which is attached at the front motor send position in terms of voltage. Analog digital converter at the PIC converts data to digital information which is then compared with requested position (data received from serial channel). If requested position is satisfied, both pins are cleared to stop. Thus, the front wheels point the right direction.

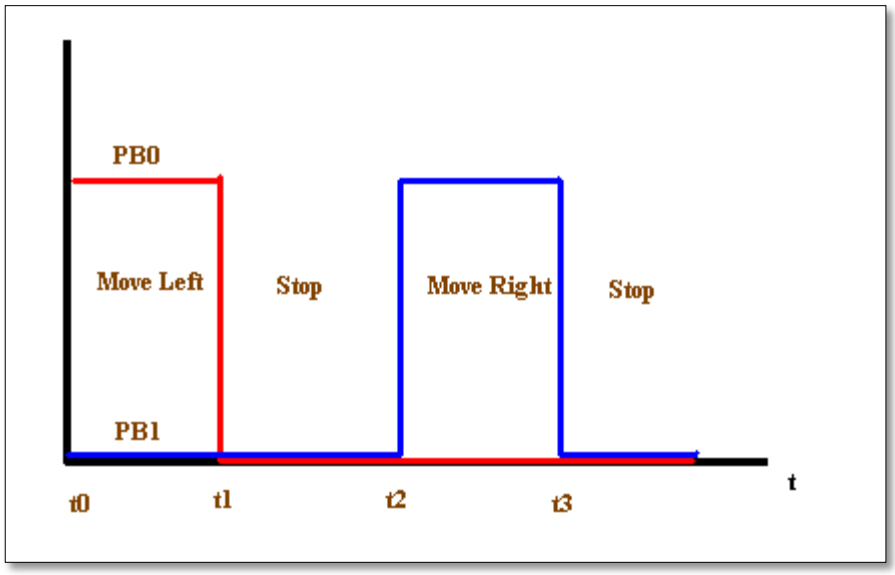


Figure IV.4 – BA6209 H-Bridge Input Signals

In Figure VI.5, We can see the rear motor circuit. We have used generic mosfet for switching issues rather than BJT transistors.

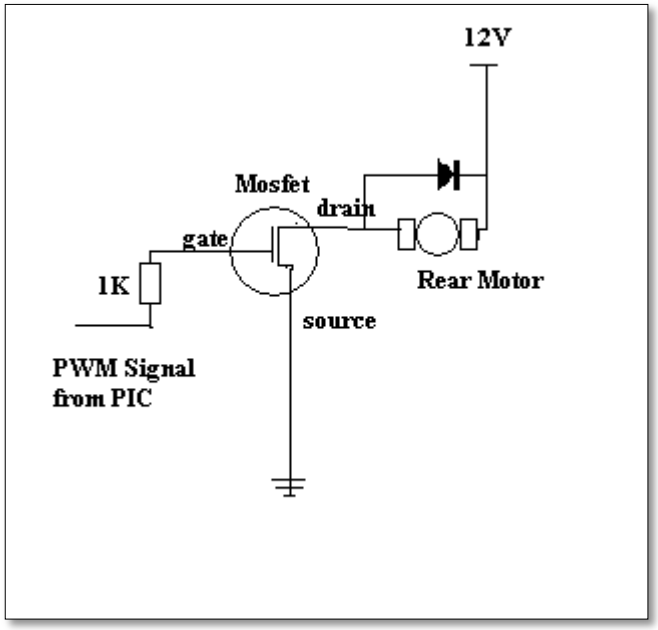


Figure IV.5 – Motor Control Circuit 2

In Figure VI.6, Rear Motor driving PWM signal example is shown. To move the vehicle faster we create PWM signal to have more busy time than idle time. To move in full speed, the signal is set mode, whereas to stop the vehicle we create a signal in all reset mode.

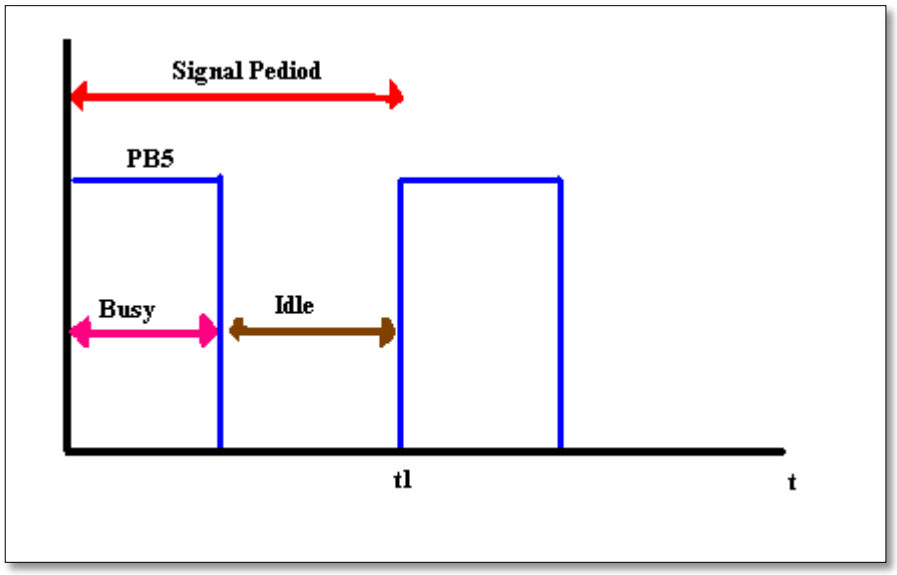


Figure IV.6 Rear Motor control PWM signal

CHAPTER VII

RESULTS AND DISCUSSION

Tracking a single object of a predefined color has been implemented in the scope of this thesis. Using a USB video input camera has increased the speed of frames that have been captured from the source, since USB bus supports high speed trasfers.

Linux operating system has provided limitless programming environment since it has GNU License with which it is able to see source code of several applications, kernel, and implement various library facilities and find various documents related to the usage of the API interaces.

Tracking algorithm has been developed in C programming enviroment which is surely the best environment for future studies. Kernel level V4L2 API interface has been used to query and grab frames from the streaming video input device. V4L2 is a Linux kernel layer for video implementation to support various video devices, thus related drivers by using common function calls and data structures.

In application level, autonomous tracking algorithms have been implemented under different light conditions. The object has been chosen to be a red triangle within a cyan rectangle. Initially, object has been held against the video input streaming camera to locate its position within the frame. Extreme light conditions, for example, reflections result in error detecting triangle. Under dark light conditions, the exposure time of camera increases dramatically which results in motion blur. To overcome motion blur, more functional cameras may be used.

CHAPTER VIII

CONCLUDING REMARKS AND RECOMMENDATION

Triangle detection is one of the difficult tasks in computer vision. Due to being arbitrary, triangle selection of interest may be compuationally high. In nature, several possible combination of edges may result in triangles. Selecting one of interest requires more criteria, such as color information in our case. This aproach dramatically reduces the computation time and increased robustness.

One major problem to this piece of work is, as decribed in previous sections, is surface reflection. Reflection of objects including the one of our interest creates false color information to the camera which creates false evaluation and error. Other problem is motion blur which creates false colors and size of the triangle detected.

We are leaving this problems to another work, since they are both subjects of heavy calculations and different aproaches. Further studies can be solving these problems to provide more robust tracking.

REFERENCES

[1] Christoph Gustav Keller, Christoph Sprunk, Claus Bahlmann, Jan Giebel, and Gregory Baratoff. Real-time recognition of U.S. Speed signs. In IEEE IV'08, 2008.

[2] A Fast Radial Symmetry Transform for Detecting Points of Interest, Loy G., Zelinsky A, IEEE PAMI, Vol. 25, No.8, August 2003. pp 959-973.

[3] P.Viola and M. Jones. Robust Real-time Object Detection. Technical report, Compaq Cambridge Research Laboratory, Feb. 2001, CRL 2001/

[4] Detection for Triangle Traffic Sign Based on Neural Network, Zhu Shuang-dong, Zhang Yi, and Lu Xiao-feng, ISNN(2), 2006, pp:40-45.

[5] Triangle Detection Based on Windowed Hough Transform, Jiang-Ping He, Yan Ma, Proceedings of the 2009 International Conference on Wavelet Analysis and Pattern Recognition, Baoding, 2009.

[6] T. Garlipp and C.H. Müller., Detection of linear and circular shapes in image analysis, Computational statistics & data analysis, Vol 51, No. 3, pp. 1479-1490, 2006.

[7] Haibing Li, Weidong Yi, An Effective Algorithm to Detect Triangles in Image, Journal of Image and Graphics, Vol 13, No.3, pp. 456-460, 2008.

[8] Canny J., A Computational Approach to Edge Detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-8, No. 7, November 1986.

[9] Fisher, W.D, On Grouping For Maximum Homogeneity, Journal of American Statistical Association, Vol. 53 (1958), pp. 789-798.

[10] Marenco D., Fortana D., Ghisio G., Monchiero G., Cardarelli E., Medici P., Porta P. P., A Validation tool for Traffic Signs Recognition Systems, Proceedings of the 12th International IEEE Conference on Intelligent Transportation Systems, St. Louis, MO, USA, October 3-7, 2009Fisher, W.D, On Grouping For Maximum Homogeneity, Journal of American Statistical Association, Vol. 53 (1958), pp. 789-798.

[11] http://en.wikipedia.org/wiki/Serial_communication (2010)

[12] http://viz.aset.psu.edu/gho/sem_notes/color_2d/html/primary_systems.html (2010)

[13] Dirks, B.: "Video For Linux Two –Devices", http://www.thedirks.org/v4l2/v4l2dev.htm (12.09.1999).

[14] http://en.wikipedia.org/wiki/YUV (10.10.2011)

[15] Kutami, A.; Maruya, Y.; Takahasi, H.; Okuno, A.: "Visual Navigation of Autonomous On-Road Vehicle", **(1990).**

[16] Dellaert, F.; Pomerleau, D.; Thorpe, C.: "Model-Based Car Tracking Integrated with a Road-Follower", **(1998).**

[17] Donhui, W.; Xiuqin, Y.; Weikang, G.: "Tracking Vechicles in Image Sequence for Avoiding Obstacles", **(1999).**

[18] Tans, S. L.; Gu, J.: "Investigation of Trajectory Tracking Control Algorithms for Autonomous Mobile Platform: Theory and Simulation", **(2005).**

[19] Jia, Z.; Balasuriya, A.; Challa, S.: "Visual 3D Target Tracking for Autonomous Vehicles", **(2004).**

[20] Jia, Z.; Balasuriya, A.; Challa, S.: "Recent Developments in Vision Based Target Tracking for Autonomous Vehicles Navigation", **(2006).**

[21] Jones, M. J.; Rehg, J. M.: "Statistical Color Models with Application to Skin Detection", *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on* **(1999).**

[22] Moon, H.; Chellappa, R.; Rosenfeld, A.: "Optimal Edge-Based Shape Detection", **(2002).**

[23] Fleyeh, H.: "Color Detection and Segmentation for Road and Traffic Signs", **(2004).**

[24] Zang, M.; Gao, W.: "An Adaptive Skin Color Detection Algorithm with Confusing Backgrounds Elimination", **(2005).**

[25] Hyun-Chul Do; Ju-Yeun You; Sung-II Chien: "Skin Color Detection through Estimation and Conversion of Illuminant Color Using Sclera Region of Eye under Varying Illumination", **(2006).**

[26] Toledo, F. J.; Martinez, J. J.; Garrigos, J.; Ferrandez, J.; Rodellar, V.: "Skin Color Detection for Real Time Mobile Applications", **(2006).**

[27] Hyun-Chul Do; Ju-Yeun You; Sung-II Chien: "Skin Color Detection through Estimation and Conversion of Illuminant Color under Various Illuminations", **(2007).**

[28] Schneirderman, H.; Kanade, T.: "A Statistical Method For 3D Object Detection Applied to Faces and Cars ", **(2000).**

[29] Peterfreund, N.: "Robust Tracking of Position and Velocity with Kalman Snakes", **(1999).**

[30] Healey, J.; Binford, T.O.: "A Color Metric for Computer Vision", **(1988).**

[31] Comaniciu, D.; Ramesh, V.; Meer, P.: "Kernel-based Object Tracking", **(2003).**

[32] Terzopoulos, D.; Szeliski, R.: "Tracking with Kalman Snakes", *Active Vision ISBN: 0-262-02351-2,* **(1993)** 3-20.

[33] Do Hyoung Kim; Do-Yoon Kim; Myung Jin Chung: "2004", *Control Automation, Robotics and Vision Conference, 2004, ICARCV 2004 8th* , Volume 2 **(2003)** 1553-1558.

[34] Aggarwal, J.K.: "Understanding of Human Motion, Actions and Interactions", *Advanced Video and Signal Based Surveillance, 2005. AVSS 2005. IEEE Conference* **(2005)** 299.