Ping-pong Detection System

Team 2
Christopher Guinnup, Rui Xu
Sam Nixon, Mengjiao Han, Bryan Schmier

Client: Scott Kay
Professor: Stephen Seigel

December 4, 2014

Contents

1 Preliminaries **3**
- 1.1 Change Log . . . 3
- 1.2 Introduction . . . 4

2 Requirements **5**
- 2.1 Stakeholders . . . 5
- 2.2 Scope . . . 5
- 2.3 Problem Domain . . . 5
- 2.4 Console . . . 6
- 2.5 Video recording . . . 7
- 2.6 General . . . 7
- 2.7 Entity-Relationship Diagram . . . 9
- 2.8 Data Flow Diagram . . . 10
- 2.9 Non-functional Requirements . . . 11

3 Design **12**
- 3.1 Architecture . . . 12
- 3.2 Design Rationale . . . 13
- 3.3 Detailed Design . . . 14
  - 3.3.1 PDSMain . . . 14
  - 3.3.2 input . . . 14
  - 3.3.3 videoService . . . 14
  - 3.3.4 computerVisionService . . . 15
  - 3.3.5 computerVisionStores . . . 16
  - 3.3.6 situationAnalysis . . . 17
  - 3.3.7 output . . . 18

4 Verification Plan **19**
- 4.1 Verification Strategy . . . 19
  - 4.1.1 Black-box techniques . . . 19
  - 4.1.2 Video Tests . . . 20
  - 4.1.3 White-box techniques . . . 20
  - 4.1.4 Testing at different levels . . . 20
- 4.2 Challenges for testing . . . 21
- 4.3 Status of testing . . . 22
  - 4.3.1 The status of black-box testing . . . 22
  - 4.3.2 The status of unit testing . . . 23

A Glossary **24**

B Bibliography **25**

Chapter 1

Preliminaries

1.1 Change Log

Date	Changes	Author	Version
9/8/14	Created document	Rui Xu Mengjiao Han	0.1
9/16/14	1. Add EDR, DFD and the descriptions of them	Bryan Schmier	0.1
9/19/14	1. Edit ERD, DFD 2. Edit grammar mistakes	Bryan Schmier	0.1
10/1/14	1. Edit Overview 2. Add Stakeholders 3. Edit ERD, DFD and their descriptions 4. Add Scope section 5. Add Problem domain section 6. Edit requirement section 7. Add non-functional requirement section	Rui Xu Mengjiao Han Bryan Schmier	0.2
10/3/14	1. Add the description of diagram 2. Change grammar mistakes	Bryan Schmier	0.2
10/3/14	1. Added Design section 2. Many editorial changes	Christopher Guinnup	0.2
10/17/14	1. Changed Design section 2. Many editorial changes 3.Add verification plan	Christopher Guinnup Byran Mengjiao Han	0.3
10/31/14	1. Major update to Design section	Christopher Guinnup	0.4
10/31/14	1. Update to Verification section	Mengjiao Han	0.4
11/7/2014	1. Design: Added `computerVisionService` submodules, `PDSMain`, cleaned up bad LaTeX syntax, positioned diagrams more logically.	Christopher Guinnup	0.5
11/14/2014	1. Verification: Added Video Tests section and performed misc proofreading and editing.	Christopher Guinnup	1.0

12/4/2014	1. Design: Changed all names to avoid abbreviations and added the new Racket modules, updated the diagrams, and misc revisions.	Christopher Guinnup	1.1

1.2 Introduction

At eMoney Company, there are several options when taking a break and relaxing with co-workers, but playing ping pong is the most popular one. The table is usually in high demand and thus results in long wait times. Workers need to check the status of the ping pong table to know whether the table is available. Moreover, the placement of this table will change sometimes. The purpose of this project is to automatically determine the status of the ping pong table by monitoring the table with a camera, and sending the availability status to employees by an external broadcast mechanism. This will prevent workers from walking to the recreation room in hopes of playing ping pong, only to find it already occupied.

Our project's name is Ping-Pong Detection System (PDS). Since the network video stream is not accessible outside of eMoney, PDS will use VLC media player(VLC) [1] to load videos already stored on a hard drive that have been recorded by the camera in the eMoney office. Since the project is part of a whole system, the output will be messages texted in console. eMoney will make more extensions based on PDS's results. Our team's name is "The Twisted Platypi" and our members are Christopher Guinnup, Sam Nixon, Bryan Schmier, Mengjiao Han and Rui Xu. The URL of our team is `http://cisc475-2.cis.udel.edu/`. We use SVN as our version control tool, and the full repository is at `svn://cisc475-2.cis.udel.edu/twistpong`. We also use Trac to track and manage the project's tasks and issues. The main page of Trac is `http://cisc475-2.cis.udel.edu/trac/twistpong/`. And the documents in the project will be prepared by LaTeX.

Chapter 2

Requirements

2.1 Stakeholders

(1) Employees of eMoney: Need a notification to let them know whether the ping pong table is available for play or not. They would also prefer not to be misled by a false result from our system. Some employees may not like the idea of software analyzing recreation room video, and may try to turn the camera away.
(2) Client Scott Kay: Would like us to deliver the software on time and in a way that fits his company's setup.

2.2 Scope

We will design a program that takes video as input, which includes recordings of the table during different states of vacancy, and then output the status of the table. Our program will rely on a library (currently the VLC media player library) for receiving video and decoding the compressed format. We have been provided with sample videos which were filmed at the eMoney office. In these videos, the angle of the camera, the status of the table (in use or not), and even the color of the ball are all different. Our program's goal is to handle these different conditions, and output the status of the ping pong table. The output will be textual, to a single console, and includes (1) "Ping pong game is in progress", (2) "The ping pong table is available", (3) "Camera no longer pointing at the table". The client will integrate this output into a broadcast system of their own.

2.3 Problem Domain

(1) Console: Will display the result of the PDS. As PDS is part of a whole system at eMoney, they will make extensions based on these results, like linking to Twitter.
(2) Table: The ping pong table which eMoney employees play on. The table is a regular dark green ping pong table with white edges, which has two sides divided by a net. The table could be moved, or it may be partially obscured by players at certain times.
(3) Camera: The input device; it records clear video of the table and transfers the video to the PDS to analyze. There are different angles between the camera and the table. The captures of videos are in Figure 2.1. Ideally, the camera is shooting vertically at the table. And in this case, the video's screen will all fill with the ping pong table. However, sometimes the camera could be at a different angle. In this case, the video's screen will only fill some part of the table.

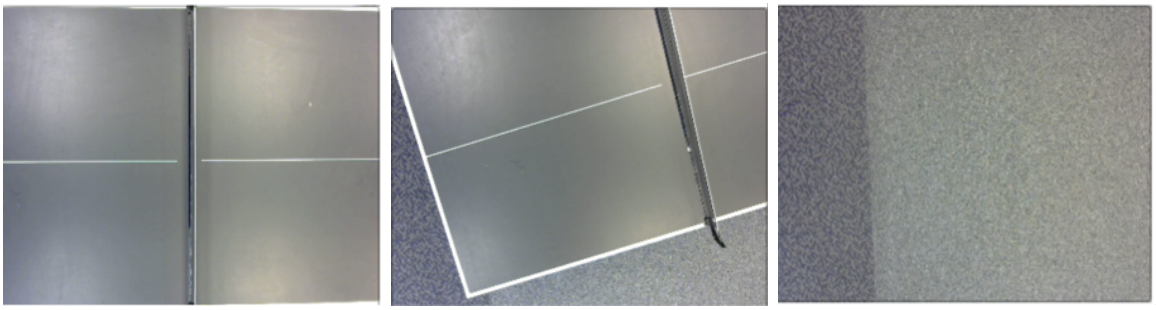


Figure 2.1: The captures of videos. (a) the ideal place of camera; (b) the different angle of camera; (c) the image when there is not a table.

(4) Ball: The ping pong ball used by employees on the ping pong table. The ping pong ball is a regular ping pong ball and the color should not affect the result (although the color will most likely be white or orange).
(5) Paddle: The ping pong paddle used for playing ping pong.
(6) Lights: Lighting around the table is an average indoor situation, approximately 50,000 lumens for the 2,700 square foot room.
(7) Playing environment: The environment around the ping pong table and tools for playing, including lighting, table, and ball.
(8) Video: Frames of data recorded by the camera.

2.4 Console

In this section we will explain in detail the requirements of each different situation.

CS1. When user input the command to close the system, then the console will shut down.

CS2. While the system is running, the console will periodically update with one of three messages and a current time stamp. The messages are: 1) "Ping pong game is in progress"; 2) "The ping pong table is available"; 3) "Camera no longer pointing at the table".

CS3. If no ball has moved for over 15 seconds, the console will display the message "The ping pong table is available".

CS4. If a ball is moving across the table, the console will display the message "Ping pong game is in progress". During this state, the ball may go off screen, off the table, or become still on the table. But as long as the ball stops for less than 15 seconds, the game is still considered in progress.

CS5. If the camera cannot detect the ping pong table, the console will text message "Camera no longer pointing at the table".

CS6. If people are standing at the ping pong table but no balls are moving and no rackets are being waved around, the console will print message "The ping pong table is available".

CS7. If people are standing at the ping pong table and waving rackets, the console will print message "Ping pong game is in progress". This status lasts until 5 seconds after they stop waving rackets.

CS8. When play ends, people may take the ball away or leave it on the table. No matter which situation appears, the result should still denote that the table is available.

CS9. When the state of the table or game has changed, the appropriate message will be printed as soon as this is detected. If the state has not changed, any repeated message will be displayed

after at least 15 seconds have passed.

2.5 Video recording

In this section we will explain in detail the requirements related to video recording.

VR1. When the table is placed horizontally, the PDS should support all camera angles from 30 to 150 degrees, as Figure 2.2 .

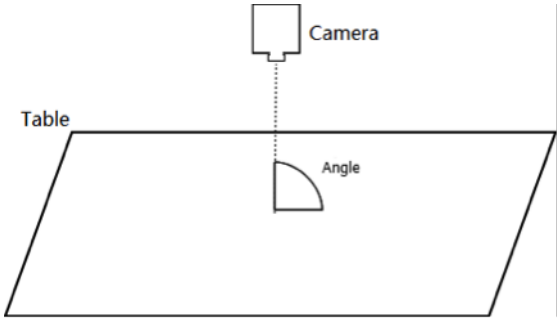


Figure 2.2: angles of the camera

VR2. When the table is rotated 90 degrees to face vertically, the PDS should support all camera angles from 30 to 150 degrees, as Figure 2.3.

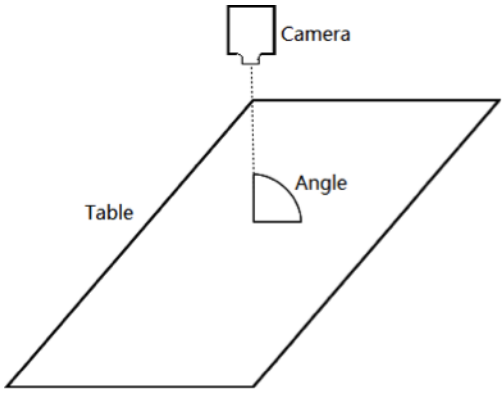


Figure 2.3: angles of the camera

VR3. MPEG2 and MPEG4 videos compression formats should be supported by the PDS.

VR4. The table could be moved, so not every video will have an entire ping pong table, as Figure 2.1(c) shows. If at least half the area of the table is in the video frame then the application should still function.

2.6 General

In this section we will explain the requirements related to the system in general.

G1. The application must have a very minimal error-rate (less than 10 percent), so that the employees can have an accurate and trustworthy indication of the table's availability.

G2. The system must be able to perform in real-time on modern computer hardware without a GPU.

2.7 Entity-Relationship Diagram

In this section, we will describe the entity-relationship diagram of the project.

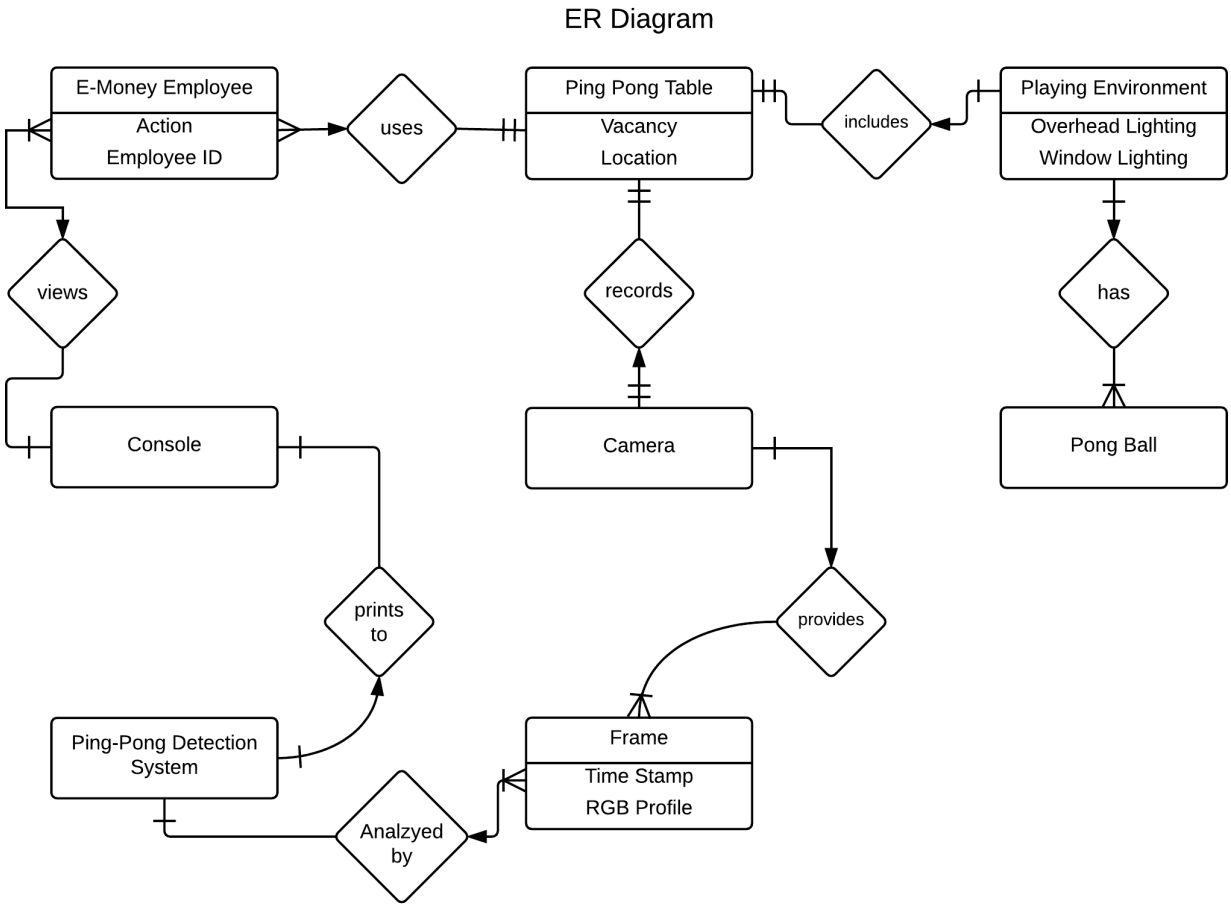


Figure 2.4: Entity-Relationship Diagram

The ER Diagram shows each entity in our working system. Each eMoney Employee is always acting, whether its playing ping pong, checking the status of the table, or doing work. At the same time, the pong table with the factors of the environment are recorded by the camera which, in turn, creates a video feed for our program, the Ping-Pong Detection System (PDS). PDS then analyzes the video feed and sends the current availability status of the table to the console.

2.8 Data Flow Diagram

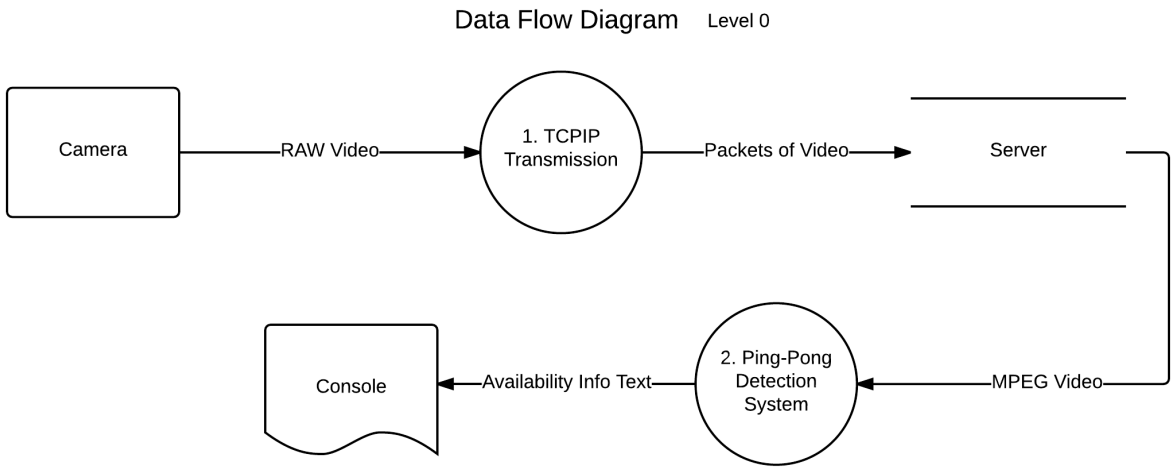
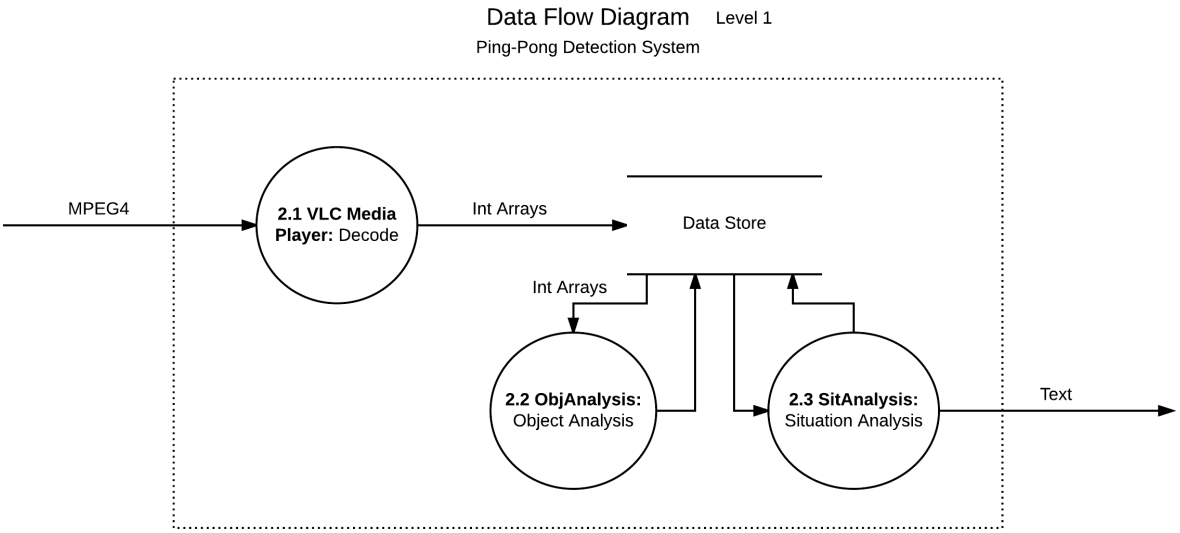


Figure 2.5: Data Flow Diagram: Level 0

The data in our system begins with the camera, which transmits recorded frames of video to the server that are relayed to the Ping-Pong Detection System. Once the video feed is analyzed by PDS, availability information text is sent to the console as final output.



Data Flow Diagram: Level 1

Taking a closer look at the Ping-Pong Detection System process, we see that the frames of video are first decoded and then stored. The frames are then analyzed by the system for object analysis and stored again. Lastly, the system performs a situation analysis and once again stores the output, as well as sending the availability information text to the console.

2.9 Non-functional Requirements

The computer which runs our Ping-Pong Detection System (PDS) must have the following:
(1) Java 1.8 or greater
(2) VLC media player 2.1.x or greater (for 32 or for 64 bit systems, whichever matches the installed JVM)
(3) OpenCV libraries 2.4.9 or greater
(4) Access to the webcam transmitting video

Chapter 3

Design

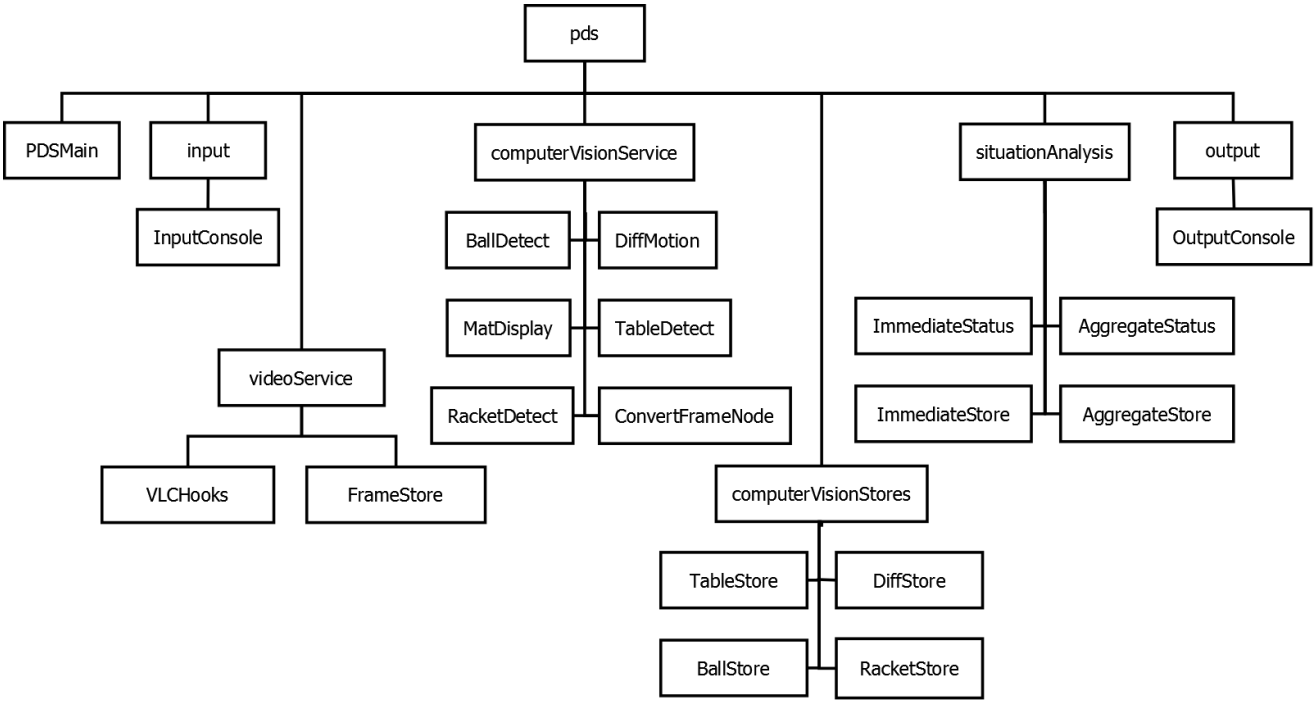


Figure 3.1: COMPRISES Diagram

3.1 Architecture

`PDSMain`: This is the main program loop for the PDS program. It initializes all application threads, determines which jobs to launch for each frame, and ensures their synchronization where necessary.
Uses: all

`videoService`: Decodes the video and provides frame data to the rest of the program.
Uses: none

`computerVisionService`: Attempts to detect motion and objects (such as the table, ping pong

ball, and ping pong rackets). Provides locations to the program.
Uses: `videoService, computerVisionStores`

`computerVisionStores`: Provides buffers which store `computerVisionService`'s location data.
Uses: none

`situationAnalysis`: Analyzes the situation from object position data provided by `computerVisionStores` and decides what the ping pong table's current status is.
Uses: `computerVisionStores, output`

`output`: Called by `situationAnalysis`, this module displays the ping-pong table status to the console.
Uses: none

`input`: Receives a user's quit command and triggers a graceful exit.
Uses: none

3.2 Design Rationale

One thread will always be checking user `input` for a quit command, so this deserves its own module. This allows changes to the way we perform the quit (for example, from a looping conditional to a message handler).

Video decoding is a very different task domain, and so benefits being seperated into module `videoService`. We also want our modules to perform comparisons between frames, for example, if calculating frame difference turns out to be a critical performance optimization. Therefore `FrameStore` is its own submodule in `videoService` so the program can access frames after they've left the decoding buffer.

The team may also decide to switch what algorithms are used for tracking and detecting objects, so this makes `computerVisionService` appealing to break into its own module. Similarly, deciding if the ping-pong table is in use or not requires its own set of algorithms and tweakable parameters. Thus `situtationAnalysis` is its own seperate component. The `computerVisionStores` module exists to give a common interface `&` storage format for these modules' exchange of information. Lastly, `output` is partitioned into its own module since it allows modifying specific wording of the output or the frequency without affecting our decision-making system.

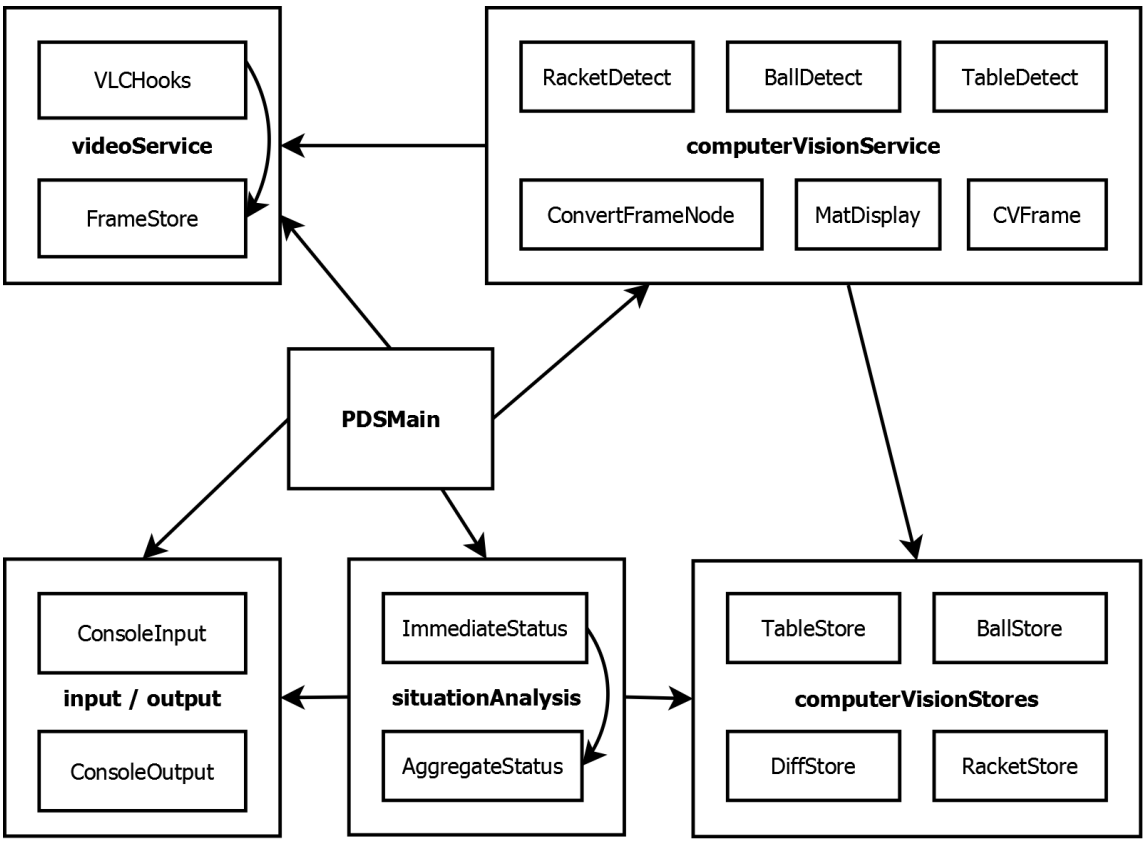


Figure 3.2: USES Diagram

3.3 Detailed Design

3.3.1 PDSMain

This module is the thread controller, which consists of the main program loop and launches image processing tasks. It uses `CountDownLatch`es to coordinate threads, and keeps the most recent two frames so that it can easily pass them to frame-comparative jobs.

3.3.2 input

Receives input from a user looking to terminate the program and triggers a graceful exit. If the user enters input that is not the quit command, `input` will inform the user of the proper quit command. This thread will run throughout the application's execution. Component `main` will check the termination status periodically.

Exports:
`boolean noQuitCommand()`

3.3.3 videoService

Hooks into the provided video decoding library and provides video frame data to the rest of the program. The functionality hides behind the interface `VideoServiceIF`. To decode video, the pro-

gram currently uses software called VLC Player through the access library VLCJ.

Exports:
`void startVideoService()`
`FrameStore getStorage()`

FrameStore

Storage and retrieval of raw frame data occurs here. To reduce latency, the buffer size has a maximum threshold after which older frames will be dropped. Frames are provided as arrays rather than pixel-by-pixel for performance reasons. An estimated 1 million pixel function calls for every still image in the video stream would severely impact real-time performance. If no frames are available, null will be returned instead.

Exports:
`FrameNode getFrame()`

FrameNode

This is the container for frame data, storing the aforementioned pixel array, the ID number of the frame, and the dimensions in pixels. Frame numbers are stores as `long` to ensure the frame number will not wrap around to a negative number (not anytime in the creators' lifetimes, at least).

Exports:
`int[] getData()`
`long getFrameNumber()`
`int getCols()`
`int getRows()`

3.3.4 computerVisionService

Attempts to detect objects such as the table, ping pong ball, and ping pong rackets. Currently we use the OpenCV library to accomplish this. Each computer vision algorithm can also run in its own thread for performance optimization. Thus inputs and outputs are passed as construction parameters, and each class is `Runnable`.

ConvertFrameNode

This class converts frames from the VLC player-decoded format to the `Mat` format which OpenCV requires as input. Depending on the needs of the computer vision algorithms, it can convert to color, grayscale, or black & white. It can also convert a frame to a binary edge-detected graphic.

Exports:
`static Mat convertFrameNodeToCVi_GRAY(FrameNode input)`

DifferenceMotion

`DifferenceMotion` calculates the overall motion between two frames. It does so by summing the absolute average difference between pixels `x`, `y` in frame 1 and pixels `x`, `y` in frame 2.

Exports:
`DifferenceMotion(CVFrame frame1, CVFrame frame2, DifferenceStore output, CountDownLatch semaphore)`

TableDetect

Tries to detect solid quadrilaterals within the image, with the goal of using these quadrilaterals to detect the presence or lack of the ping-pong table. It uses a series of thresholds, magnifications, dialations, and edge detections from OpenCV.

Exports:
`TableDetect(CVFrame frame, TableStore output, CountDownLatch semaphore)`

BallDetect

Attempts to detect the ball travelling over the table, apart from any other motion occuring in the video.

Exports:
`BallDetect(CVFrame frame1, CVFrame frame2, BallStore output, CountDownLatch semaphore)`

RacketDetect

Uses a reference racket shape to fit any objects in the image. To speed comparisons, the video image is first filtered by various OpenCV methods, reducing the candidate objects to be compared.

3.3.5 computerVisionStores

Provides locations and other per-frame or per-frame-comparison data (such as summed frame difference). Consists of a store for each kind of visual analysis performed on the video. Each store possesses a buffer with a constant `CAPACITY`, and when this is exceeded old values are dropped. This is to save memory yet still allow the program to smooth sudden abberations in detection results.

DifferenceStore

Gives access to the computed frame-difference values.

Exports:
```
long getDifference(long frameNum)
boolean wasSkipped(long frameNum)
```

TableStore

Stores the quadrilaterals which are potentially from lines on the table.

Exports:
```
ArrayList<MatOfPoint2f> getTableResults(long frameNum)
boolean wasSkipped(long frameNum)
```

BallStore

Stores the potential ball candidates found with `BallDetect`.

Exports:
`List<RotatedRect> getBoundingRectangles(Long frameNum)`

RacketStore

Stores the best racket-shape fit (0 is an exact fit, greater numbers deviate more) determined from each video frame.

Exports:
`double getBestShapeFit(long frameNum)`

3.3.6 situationAnalysis

Analyzes the situation from object data provided by `computerVisionStores` and decides what the ping pong table's current status is. This module looks not only at the immediate results, but also at an aggregation of the data. This way a singular erroneous result should be smoothed out by taking the rest of the data into account. Upon detecting a true change in status, it notifies the `output` module.

ImmediateStatus

This module converts the object data from `computerVisionStores` into a rough probability that a certain state is true (such as: the table is within the camera's view). It then inserts this into an `ImmediateStore` buffer which it passes to `AggregateStatus` to perform a longer-range smoothed analysis. `ImmediateStatus` itself runs as a thread so that these analyses do not delay the decoding and extraction of new video frames.

Exports:
```
ImmediateStatus(long frameNum, CVStores input, StatusStores output)
void run()
```

ImmediateStore

Stores the rough probabilities calculated by `ImmediateStatus` and provides them to `AggregateStatus`. The node class which it returns contains probabilities for the states that: 1) the table is available, 2) the table is in use, 3) the table is in the video frame, 4) there is motion in the video, and 5) the camera is encountering an error.

Exports:
```
ImmediateNode getProbabilities(long frameNum)
ImmediateNode [] getMultipleProbabilities(int count)
int getSize()
```

AggregateStatus

Uses `ImmediateStore` (and potentially `computerVisionStore` in the future, if an edge case might arise) to determine the smoothed status over a length of several frames. (The exact number of frames is set as a constant `SMOOTHING_FACTOR`.) The results are then placed in `AggregateStore` so that the main loop may access the results, which it needs in determining which computer vision jobs to launch.

Exports:
`void parseImmedStatus()`

AggregateStore

Stores the final decisions made as to the status of the table. Also includes variables which specifically store the last status which was decided, and the last frame that each status was detected. Makes values available to the main loop and for future status calculations.

Exports:
`enum GameStatus getGameStatus()`
`long getStatusFrameNum(enum GameStatus)`

3.3.7 output

Display simply reads the current status of the ping pong table and outputs it to the console. Exact wording, insertion of a timestamp, and the method of output (console) are all controlled here.

Exports:
`void printStatus(enum status)`

Chapter 4

Verification Plan

4.1 Verification Strategy

4.1.1 Black-box techniques

Requirements	Tests
CS1	ConsoleInputTest
CS2	ConsoleOutputTest
CS3	TableAvailableTest
CS6	TableAvailableTest
CS4	GameProcessTest
CS7	GameProcessTest
CS5	NoTableTest
CS9	RealTimeTest
G1	RealTimeTest
VR3	VideoServiceTest

Table 4.1: The tests for requirements

ConsoleInputTest: The ConsoleInputTest is used to test the requirement of CS1, the input of the console. When the user inputs the command to quit, the console will shut down.

ConsoleOutputTest: The ConsoleInputTest is used to test the requirement of CS2, the output of the console.

TableAvailableTest: Testing the requirement CS2 and CS5, which stand for the condition that ping pong table is available. When the testing results of these two requirements are "The ping pong table is available",the tests are passed.

GameProcessTest: Testing the requirement CS3 and CS5, which are the situations that ping pong game is in process. When the testing results of CS3 and CS5 are "Ping pong game is in process", the tests are passed.

NoTableTest: NoTableTest is the testing for CS4, in which condition there no longer exist a

table in the video. When the testing result of CS4 is "Camera is no longer pointing at the table", the test is passed.

RealTimeTest: RealTimeTest is used for testing the ability of real time of the project, as the requirements of CS8 and G1. When the project can change the appropriate message as soon as the new statue is detected, the test is passed.

VideoServiceTest: VideoServiceTest is used to test the performance of video decoding.

4.1.2 Video Tests

Due to the application's real-time nature, much testing will necessarily involve non-automated processes. For one, automated video testing would increase the time to perform tests by several minutes at each commit, since the application decodes strictly in real-time and uses interval-based smoothing to iron out anamalous signals (such as a video frame corrupted in transmission). Therefore, several test videos have been made from the samples provided by Scott Kay. These are available at `http://cisc475-2.cis.udel.edu/testvids/` and test the following cases:

Test Cases
Free to In-Use
In-Use with Long Gap
In-Use to Free with Rackets
No Table with Motion
Little Table with Little Motion

Table 4.2: Video Tests

4.1.3 White-box techniques

For white-box techniques, we will use Jacoco for our project. And for each module, statement coverage and branch coverage will be done. The statement coverage need to achieve 100 percent. Moreover, we expect the branch coverage to get 60 percent to 70 percent.

4.1.4 Testing at different levels

1. Unit testing plan
Unit testing plan shown in table 4.3
Please note that we are unable to write tests that pass on the headless UDel virtual machine for class `VLCHooks` due to its invocation of GUI elements necessary for VLC Player decoding interface. These same tests pass successfully in a GUI environment.
2. Integration testing plan The order of integration testing plan is shown in Figure 4.1
(1)The first two modules we need to test are VLCHooks module and FrameStore module. After they pass tests, the videoService should too be tested.
(2)After test DiffMotion, Lines, VectorMotion and Objects, these four modules can integrate to test the module of computerVisionService.

Packages	Classes	Unit Tests	Stubs
pds	PDSMain	PDSMainTest	
pds.computerVisionService	ConvertFrameNode CVStores Lines LineStore BallDetect DiffMotion DiffStore	CoverFrameNodeTest CVStoresTest LineTest	DiffMotionStub DiffStoreStub LineStoreStub
pds.input	ConsoleInput	ConsoleInputTest	
pds.output	ConsoleOutput	CosoleOutputTest	
pds.situationAnalysis	AggregateStatus AggregateStore ImmediateStatus ImmediateStore		AggregateStatusStub AggregateStoreStub ImmediateStatusStub ImmediateStoreStub
pds.videoService	FrameNode FrameStore VideoServiceIF VLCjVideoService EXLUDING VLCHooks	VideoServiceTest	FrameStoreStub

Table 4.3: Unit Testing Plan

(3)AggregateStatus and ImmediateStatus should to be tested in order before we integrate them to test situationAnalysis.
(4)computerVisionStores can be tested after all the submodules are passed.

3. System testing System testing will be done after the integration testing. The whole system shows on Figure 3.2. Make sure the whole system works, we need to test whether the system can realize the requirements, so that a lot of different videos are required.

4. Acceptance testing After the system testing is passed, we will deliver the project to the client to do the acceptance testing.

4.2 Challenges for testing

1. Testing a particular case, we need to prepare a short video for it. However, although the test is passed, we cannot sure the system can deal with the actual video. There are a lot of differences between two videos, such as the position o table or the angle of camera. Therefore, we need to prepare several different videos to test one case.

2. As we do testing on different level, there is not a actual value we can test. As unit testing 'LinesTest' for example, this class doesn't have an actual return value, it just return a line, its start point and end point. Moreover, for different video, the returns are also different. Therefore, how to test these class is a challenge.

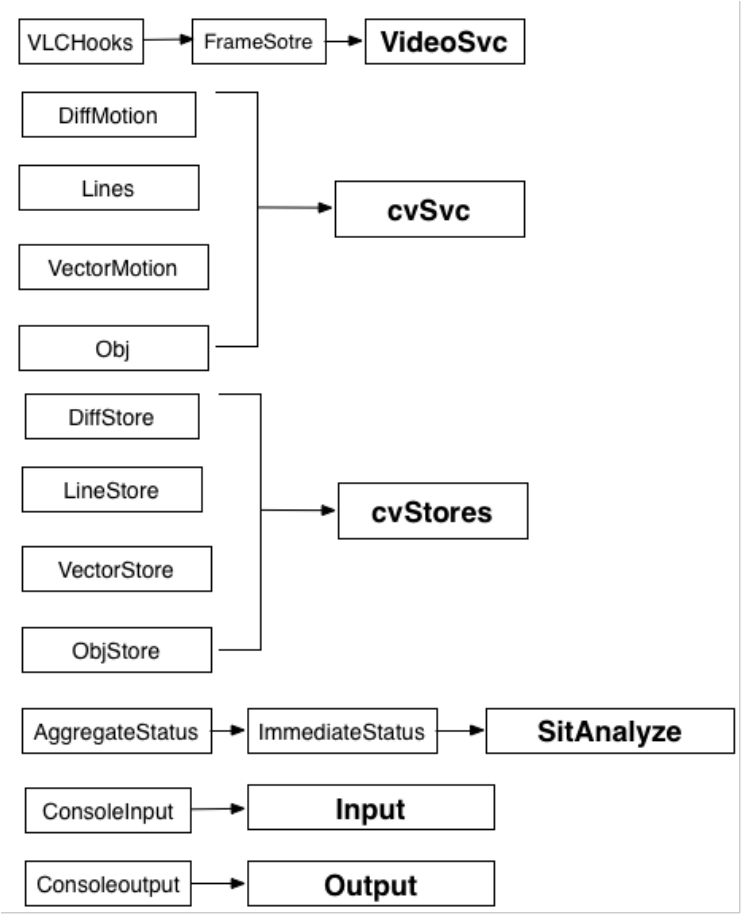


Figure 4.1: Integration Plan

4.3 Status of testing

4.3.1 The status of black-box testing

Black-box tests	Status
ConsoleInputTest	passed
ConsoleOutputTest	passed
TableAvailableTest	not written yet
GameProcessTest	not written yet
NoTableTest	not written yet
RealTimeTest	not written yet
VideoServiceTest	passed

Table 4.4: The status of Black-box testing

4.3.2 The status of unit testing

Unit tests	Status
PDSMainTest	passed*
CoverFrameNodeTest	passed
CVStoresTest	passed
LineTest	passed
ConsoleInputTest	passed
CosoleOutputTest	passed
VideoServiceTest	passed

Table 4.5: The status of unit testing

*One test in `PDSMainTest` is not passing on the UDel virtual machine due to the machine's headless nature and our program's need to initialize GUI elements, but this test does pass in a GUI environment.

Appendix A

Glossary

DFD
: Data Flow Diagrams show the flow of data through the system.

ERD
: Entity-Relationship Diagrams show relationships among entities.

frame
: A video is composed of many still images, called frames, which are displayed at a rapid rate to appear as a moving scene.

GPU
: A Graphics Processing Unit, which aids some computers in rendering 3D and 2D graphical content.

GUI
: Graphical user interface, such as Windows. This is in contrast to a textual command line terminal.

MPEG2/MPEG4
: Two related standards for lossy compression of video content.

PDS
: PDS is short for Ping-Pong Detection System, the name of the project.

VLC
: VLC media player(VLC) is a portable, free and open-source and streaming media player, which is used by the project to recieve and decode video. Importantly, VLC is capable of decoding the MPEG2/MPEG4 formats.

VLCJ
: The library of VLC for java

Appendix B

Bibliography

[1] http://vlc-player.downloadape.org/about