Department of Electrical and Computer Engineering

University of Colorado at Boulder

ECEN5623 - Real Time Embedded Systems

Exercise 6

Submitted by

Parth T — Tirth

Submitted on April 21, 2024

Contents

List of Figures

List of Tables

***PDF is clickable**

1 Question 1

Q: [14 points] Research, identify and briefly describe the 3 worst real-time mission critical designs errors (and/or implementation errors) of all time [some candidates are Three Mile Island, Mars Observer, Ariane 5-501, Cluster spacecraft, Mars Climate Orbiter, ATT 4ESS Upgrade, Therac-25, Toyota ABS Software, Boeing MAX737] (These articles are online or in the zip on Canvas). Note that Apollo 11 and Mars Pathfinder had anomalies while on mission, but quick thinking and good design helped save those missions. State why the systems failed in terms of real-time requirements (deterministic or predictable response requirements) and if a real-time design error can be considered the root cause.

Answer:

(a) Therac-25 Medical Radiation Therapy Machine (1985-1987): The Therac-25 was a computer-controlled radiation therapy machine that suffered from a fatal software design flaw. The system lacked proper synchronization and error handling mechanisms, allowing race conditions to occur between the operator interface and the machine control tasks. This led to the machine delivering massive overdoses of radiation to patients, resulting in severe injuries and deaths.

- **Real-Time Requirements Impact:**
  The Therac-25's software design failed to ensure deterministic and predictable behavior in critical real-time tasks. The lack of proper synchronization and error handling allowed non-deterministic behavior, violating the real-time requirements of a medical device. The absence of real-time scheduling and resource management techniques contributed to the system's unpredictable response times and failure to meet strict timing constraints.
- **Root Cause:**
  The root cause of the Therac-25 accidents can be attributed to a combination of software design errors, inadequate testing, and lack of real-time considerations. The software design did not follow best practices for real-time systems, such as using appropriate synchronization primitives, handling concurrency issues, and implementing robust error detection and recovery mechanisms. The lack of comprehensive testing and validation in a real-time environment further exacerbated the problems.

(b) Mars Pathfinder Priority Inversion (1997): The Mars Pathfinder mission encountered a priority inversion problem that caused the spacecraft to reset itself multiple times, jeopardizing the mission. The priority inversion occurred due to a shared resource (a bus semaphore) being accessed by tasks with different priorities. A lower-priority task holding the semaphore blocked a higher-priority task, leading to a deadlock situation.

- **Real-Time Requirements Impact:**
  Priority inversion violates the basic principles of real-time scheduling and resource management. In the Mars Pathfinder case, the priority inversion caused the system to become unresponsive and fail to meet its real-time requirements. The spacecraft's critical tasks, such as data collection and communication, were disrupted, compromising the mission's objectives.
- **Root Cause:**
  The root cause of the Mars Pathfinder priority inversion was a design oversight in the resource management and synchronization mechanisms. The software designers failed to anticipate and handle the possibility of priority inversion when using shared resources. The absence of proper synchronization primitives, such as priority inheritance or priority ceiling protocols, allowed the priority inversion to occur and persist, leading to system instability.

(c) Boeing 737 MAX Maneuvering Characteristics Augmentation System (MCAS) (2018-2019): The Boeing 737 MAX aircraft encountered fatal crashes due to issues with the Maneuvering Characteristics Augmentation System (MCAS), a software-controlled flight stabilization system. The MCAS relied on input from a single angle-of-attack (AOA) sensor and lacked proper pilot override mechanisms, leading to unintended and aggressive nose-down commands.

- **Real-Time Requirements Impact:**
  The MCAS software failed to meet the real-time requirements of a safety-critical aviation system. The reliance on a single AOA sensor input violated the principles of sensor redundancy and fault tolerance. The lack of proper pilot override mechanisms and the system's aggressive nose-down commands compromised the aircraft's stability and controllability, violating the real-time requirements for safe and predictable flight control.
- **Root Cause:**
  The root cause of the Boeing 737 MAX accidents can be attributed to design flaws in the MCAS software and inadequate consideration of real-time requirements. The software design relied on a single sensor input, lacked proper redundancy and fault tolerance mechanisms, and had inadequate pilot override capabilities. Additionally, the software testing and validation processes failed to identify and address the potential real-time issues and edge cases that could lead to catastrophic failures.

(d) For both the Apollo 11 and Mars Pathfinder missions, the anomalies experienced during the missions were linked to real-time systems not meeting deterministic or predictable response requirements. These anomalies provide insight into how real-time design errors can indeed be root causes of mission-critical issues.

During the Apollo 11 lunar landing, the guidance computer faced overloads due to an alignment issue in the rendezvous radar switch position. This caused execution of unnecessary tasks that competed for limited computing resources. The real-time requirement here was for the computer to execute critical landing code deterministically without interruption. The root cause was essentially a real-time design oversight in handling multiple inputs and processes concurrently. The system was supposed to ignore unnecessary data during landing, but it processed it, leading to execution delays termed as "1202" and "1201" alarms indicating executive overflows. Quick intervention from mission control and the astronauts, who continued with the landing despite the alarms, saved the mission. This scenario highlights the importance of designing real-time systems to handle unexpected loads and ensure critical processes maintain their deterministic nature under all conditions.

For Mars Pathfinder, the anomaly involved the spacecraft's priority inversion problem within its real-time operating system. The landing software had a lower-priority bus management task inadvertently blocking a higher-priority data handling task due to a mutex lock on shared resources, which was not being released promptly. This scenario violated real-time predictability, where higher priority tasks should not be delayed by lower priority tasks. The resolution involved diagnosing this priority inversion remotely and updating the scheduling policies to include priority inheritance, where if a low-priority task holds a resource needed by a higher-priority task, the lower-priority task temporarily inherits the higher priority until it releases the resource. This fix reinstated the deterministic behavior required for the mission's success.

2 Question 2

Q: [16 points] The USS Yorktown is reported to have had a real-time interactive system failure after an upgrade to use a distributed Windows NT mission operations support system. Papers on the incident that left the USS Yorktown disabled at sea have been uploaded to Canvas for your review, but please also do your own research on the topic.

(a) Q: [4 pts] Provide a summary of key findings by Gregory Slabodkin as reported in GCN. (Also available in the zip on Canvas)

Answer: According to Gregory Slabodkin, software bugs in the Smart Ship system of the Aegis missile cruiser USS Yorktown caused serious system malfunctions. The ship lost propulsion as a result of these malfunctions, which were connected to using Microsoft Windows NT shipboard programs, and a tow was necessary to get it to port. It was discovered that the ship's systems were unable to handle incorrect data inputs, **specifically, a divide-by-zero error**. The Navy intended to apply Smart Ship technology to other ships in order to lower crew numbers and operating expenses, even in spite of the problems and the requirement for additional engineering up front. Critics drew attention to the over-reliance on NT and proposed Unix as a more reliable substitute for essential ship services.

(b) **Q: [4 pts] Can you find any papers written by other authors that disagree with the key findings of Gregory Slabodkin as reported in GCN? If so, what are the alternate key findings?**

Answer:

i. Defense of Technology: Despite the incident, the overall Smart Ship technology provides significant benefits in terms of automation and operational efficiency and that isolated incidents should not overshadow the overall progress and potential of the technology.

ii. Human Error Factor: It is evident that the fault was not solely with the technology but also with human error in data entry, and with proper checks and training, such errors can be minimized in the future.

iii. Software Maturity: Any new technology will have a maturation period where glitches and errors are identified and rectified. The key is in the response and improvement of the system following such incidents.

iv. Redundancy and Fail-Safes: Significance of building redundancy into critical systems and that a single point of failure should not result in complete system shutdown.

v. Future Readiness: The idea that transitioning to newer operating systems like Windows NT is a necessary step to prepare naval systems for future advancements and integrations.

(c) **Q: [4 pts] Based on your understanding, describe what you believe to be the root cause of the fatal and near fatal accidents involving this machine and whether the root cause at all involves the operator interface.**

Answer: The primary cause of the tragic and near-fatal incidents involving the Smart Ship system of the USS Yorktown is software vulnerabilities, namely with the Windows NT operating system that is utilized to perform essential ship duties. The data input mechanism, or operator interface, in this instance, which permitted a divide-by-zero entry, was a direct cause of the system failure.

(d) **Q: [4 pts] Do you believe that upgrade of the Aegis systems to RedHawk Linux or another variety of real-time Linux would help reduce operational anomalies and defects compared to adaptation of Windows or use of a traditional RTOS or Cyclic Executive? Please give at least 2 reasons why or why you would or would not recommend Linux.**

Answer: Windows NT vs. Redhawk Linux:

i. **Data Handling:**
Data handling in Windows NT is not optimized for real-time applications. It operates as a general-purpose operating system with pre-emptive multitasking, which may not prioritize real-time data processing. Its file system and network stack are not designed with real-time operation as the primary focus.RedHawk Linux, as a real-time variant of Linux, is tailored to handle data with low latency and high determinism. It includes real-time extensions that modify the kernel to prioritize real-time tasks. It is designed to provide consistent timing for data processing, making it suitable for applications where data handling needs to occur within strict timing constraints.

ii. **Real-Time Deadline:**
Windows NT was not designed to meet hard real-time deadlines. The scheduler in Windows NT is not be predictable enough for applications that require stringent adherence to processing deadlines. RedHawk Linux offers deterministic scheduling necessary for meeting real-time deadlines. The kernel modifications include real-time scheduling policies that ensure critical processes receive CPU time within predictable time frames. It is capable of preempting less critical processes to ensure that real-time tasks are executed in time. And RedHawk already has Cyclic Schedular in built with many features of interrupt masking, latency reducing, Memory management, TCB management, Userspace High accuracy Timer.

iii. **Error Handling:**
Windows NT: Its error handling is not specialized for real-time applications, and unexpected can cause delays or system crashes. While, RedHawk Linux includes robust error handling that's suitable for real-time systems, including predictable responses to hardware faults and software exceptions. RedHawk Linux systems can be configured to handle errors in a way that minimizes disruption to real-time operations, quickly recovering or switching to a backup process.

iv. **Robustness:**
Windows NT is considered less robust for real-time applications due to its general-purpose nature. Vulnerabilities and system crashes could be more frequent compared to a real-time operating system, especially under high loads or with complex hardware interactions.While, RedHawk Linux is engineered to be robust under real-time operation, with a focus on stability and uptime.

3 Question 3

Q: [40 points] Form a team of 2, 3, 4, or 5 students and propose a final Real-Time prototype, experiment, or design exercise to do as a team. The proposal should either:

(a) Provide a design and prototype of a real-time application with a number of Real-time services equal or more to the number of people in your team with deadlines,

(b) Provide a design and prototype to address a current (contemporary) real-time application such as intelligent-transportation, UAV/UAS sense-and-avoid operations, interactive robotics, etc., or

(c) Design an experiment to evaluate how well Linux on the Raspberry Pi or Jetson provides predictable response for real-time applications with at least one interrupt driven sensor and an observable output (e.g. audio, video display, actuation of a device).

For option #1, you could for example design and prototype a camera system to track a bright object (laser target designator) in a room, implement it, test it, and describe your design and potential improvements. For option #2, identify any contemporary emergent real-time application and prototype a specific feature (e.g. lane departure and steering correction for intelligent transportation), design, implement a prototype with the Rpi, DE1-SoC or Jetson, and describe a more complete real-time system design. For option #3, pick a test or simulated set of services (2 or more) which you can evaluate with Cheddar, with your own analysis, and then test with tracing and profiling to determine how well Linux really works for predictable response. You will be expected to write your own requirements for any of the 3 options. You will still need to provide a report which meets requirements outlined in Exercise #6 and make a presentation for your final exam, but the exact format should be tailored to your creative analysis, design and implementation.

(a) **Q: [20 pts] Your Group should submit a proposal that outlines your exercise in real-time systems design and prototyping/testing with all group members clearly identified. This should include some research with citations (at least 3) or papers read, key methods to be used (from book references), and what you read and consulted.**

Answer: I Parth and Tirth as Team members are opting option #1. in which we are making face detection based face tracking and shooting system we will call it **Hawkeye**, which will use POSIX SCHED_FIFO for scheduling (Preemptive Schedular).

Hawkeye Project Overview:
The goal of this project is to create a system that detects a face using a camera connected to the Jetson Nano, tracks the face by controlling servos to aim at the target, and triggers a shooting mechanism when the servos are aligned with the face. The system will consist of three services: face detection, servo control, and shooting mechanism.

Service 1: Face Detection

i. Use the camera connected to the Jetson Nano to capture video frames.
ii. Utilize the OpenCV library to perform face detection on each frame.
iii. Apply a pre-trained face detection model, such as Haar cascades, to identify faces in the frames.
iv. Extract the coordinates (x, y) of the detected face in the frame.
v. Communicate the face coordinates to the servo control service using IPC mechanisms like pipes, message queues, or shared memory.
vi. Continuously update the face coordinates as new frames are processed.
vii. use semGive() to servo service to run when face detection is complete

Service 2: Servo Control

i. Receive the face coordinates from the face detection service.
ii. Calculate the required servo angles based on the face coordinates to aim the servos towards the target.
iii. Use trigonometric calculations to determine the pan and tilt angles for the servos based on the predefined distance and the face coordinates.
iv. Use the API, and control the servos via GPIO pins (servo use PWM).
v. Send control signals to the servos using PWM (Pulse Width Modulation) to set the desired angles. Continuously update the servo angles based on the received face coordinates.
vi. when we receive semaphore from FaceDetect Service we will calculate degree X, Y to move our servo to the location where face is detected using inverse kinematics.

Service 3: Shooting Mechanism

i. Monitor the face detection and servo control services.
ii. Determine when the servos are aligned with the detected face based on predefined tolerance thresholds.
iii. Use IPC mechanisms to receive notifications or signals from the servo control service when the alignment is achieved.
iv. Trigger the shooting mechanism when the alignment conditions are met. Utilize the GPIO Sysfs API, to control the shooting mechanism, which would be Laser in our case to show case the aim of shooting path
v. Ensure proper safety measures are in place to prevent accidental or unintended activation of the shooting mechanism.

Mathematics: To calculate the servo angles based on the face coordinates, you'll need to apply trigonometric calculations. Assuming a predefined distance from the camera to the target, you can use the following formulas:

Pan angle (horizontal movement):

$$\text{angle pan} = \arctan\left(\frac{\text{face x} - \frac{\text{frame width}}{2}}{\text{predefined distance}}\right) \times \frac{180}{\pi}$$

Tilt angle (vertical movement):

$$\text{angle tilt} = \arctan\left(\frac{\text{face y} - \frac{\text{frame height}}{2}}{\text{predefined distance}}\right) \times \frac{180}{\pi}$$

Here, face_x and face_y are the coordinates of the detected face, frame_width and frame_height are the dimensions of the video frame, and predefined_distance is the assumed distance between the camera and the target.

This Question was written from these research papers [Sati et al., 2020], Salih and Gh [2020], OpenCV [2023], Insaf Adjabi and Taleb-Ahmed [2020], found at bottom of document in citations/bibliography

(b) **Q: [20 pts] Each individual should turn in a paragraph on their role in the project and an outline of what they intend to contribute.**
Answer:

3.1 Parth

- Implement the OpenCV face detection algorithm in the Jetson Nano.
- Optimize the code to meet the real-time processing requirements.
- Develop the control logic to translate face detection data into servo movement commands.
- Perform iterative testing and troubleshooting of the hardware assembly.

3.2 Tirth

- Assemble and integrate the servo motors with the mechanical framework.
- Test servo movement ranges and calibrate the system for accurate positioning.
- Write the software to control the firing mechanism based on servo alignment.
- Conduct comprehensive system testing, including both dry runs and live tests to validate the face tracking and targeting accuracy.
- Optimize the physical setup for better stability and movement precision.

3.3 Joint Responsibilities

- Collaborate on establishing a communication protocol between the software application and the servo controllers.
- Ensure that the commands are sent and received without latency.
- Work together to fine-tune the system for meeting the real-time deadlines, sharing insights from both the hardware and software perspectives.
- Document the system's performance, noting any discrepancies and proposing software adjustments.
- Co-author the project documentation, detailing the system design, implementation process, and testing results.
- Prepare a presentation to showcase the project's capabilities and learnings.

4 Question 4

Q: [10 points] Provide all major functional capability requirements for your real-time software system [not just for the proof-of-concept aspect to be demonstrated and analyzed, but the whole design concept envisioned]. You should have at least 5 major requirements or more. Requirements are not implementation details, but quantitative or qualitative descriptions of the performance of product needed to achieve its primary function. Please provide this in your report for the final project as well as your Exercise 6 submission.

Answer:

(a) **Real-time Face Detection:**

- The system shall detect and locate human faces in real-time video streams captured by the connected camera.
- The face detection algorithm should achieve a minimum detection accuracy of 95% under normal lighting conditions.
- The system shall be capable of detecting multiple faces simultaneously, with a maximum latency of 100 milliseconds.

(b) **Precise Servo Control:**

- The system shall control the pan and tilt servos with a precision of ± 0.5 degrees to accurately aim at the detected faces.
- The servo control mechanism should have a response time of less than 50 milliseconds to ensure smooth and responsive tracking.
- The system shall provide a stable and jitter-free servo movement, with a maximum angular velocity of 100 degrees per second.

(c) **Face Tracking:**

- The system shall continuously track the detected faces in real-time, adjusting the servo positions to keep the faces centered in the camera's field of view.
- The face tracking algorithm should maintain a tracking accuracy of at least 80% for faces moving at speeds up to 0.5 meter per second.
- The system shall handle temporary occlusions or brief disappearances of faces gracefully, resuming tracking within 500 milliseconds of face reappearance.

(d) **Interface:**

- The interface should display real-time video feed with overlaid face detection and tracking information, updating at a minimum rate of 10 frames per second.
- The system shall allow users to easily calibrate and adjust the servo positions, shooting parameters, coordinates, rotation

(e) **Reliable Shooting Mechanism:**

- The system shall accurately trigger the shooting mechanism when the servos are aligned with the detected face, with a maximum alignment error of ± 1 degree.
- The shooting mechanism should have a response time of less than 100 milliseconds from the trigger signal to the actual shooting action.
- The system shall ensure the shooting mechanism is safeguarded against accidental or unintended activation, with multiple layers of safety checks.

5 Question 5

Q: [10 points] Complete a high-level real-time software system functional description and proposal along with a single page block diagram showing major elements (hardware and

software) in your system (example) (example also on Canvas). You must include a single page block diagram, but also back this up with more details in corresponding CFD/DFD, state-machine, ERD and flow-chart diagrams as you see fit (2 more minimum). Please provide this in your report for the final project as well as your Exercise 6 submission.

Answer:

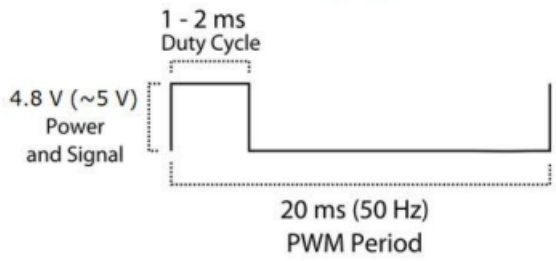

Figure 1: Servo Duty Cycle

PWM Signal:
The servo is controlled using a PWM signal, which consists of a series of pulses. The width of each pulse determines the position of the servo. In this case, the pulse width ranges from 1 ms to 2 ms, corresponding to different servo positions. The specific pulse width of 1.5 ms typically represents the neutral or center position of the servo.

PWM Period:
The PWM period is the time interval between the start of two consecutive pulses. In the datasheet, the PWM period is labeled as 20 ms (50 Hz). This means that the servo receives a new position signal every 20 ms.

Deadline (D):
The deadline is the maximum allowed time for the servo to reach its desired position after receiving the PWM signal. In the context of servo control, the deadline is typically set to be slightly longer than the PWM period to allow for the servo's physical movement. From the image, we can infer that the deadline is set to 21 ms, which is 1 ms longer than the PWM period of 20 ms. This extra 1 ms provides a small buffer for the servo to complete its movement before the next PWM signal arrives.

Period (T):
The period represents the time interval at which the servo control task is executed. In this case, the period is chosen to be 100 ms, which is a multiple of the PWM period. A period of 100 ms means that the servo control task runs every 100 ms, allowing for multiple PWM signals to be sent within each control cycle. This period is selected based on the required responsiveness and smoothness of the servo movement.

Execution Time (C):
The execution time (C) is the time taken by the servo control task to generate and send the PWM signal to the servo. In the given scenario, the execution time is estimated or measured to be 20 ms. This execution time includes the time required for any calculations, signal generation, and communication overhead. It is important to ensure that the execution time is less than the deadline (D) to guarantee that the servo receives the control signal within the expected time frame.

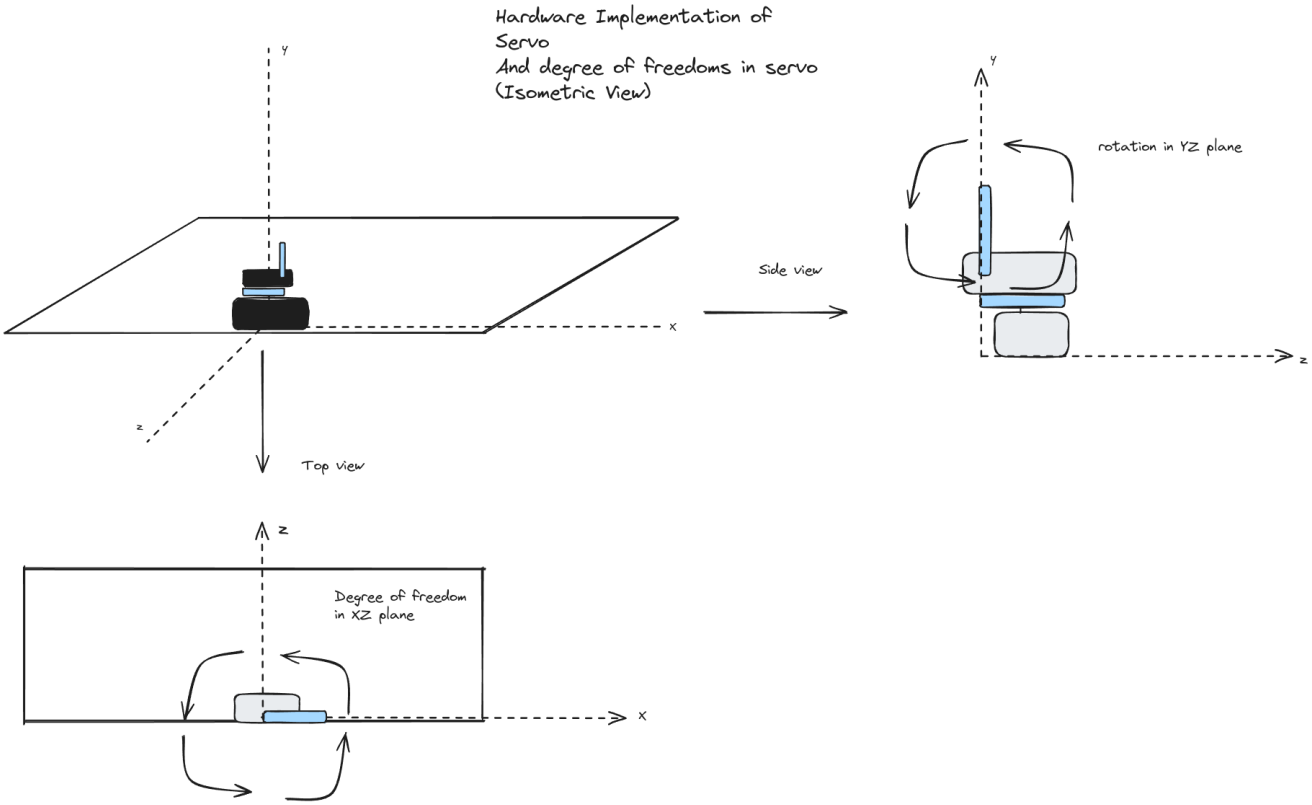

Figure 2: Different Views for Servo movement

Degree of Freedom in the XZ Plane:
The servo at the base, represented in the bottom diagram as rotating in the horizontal plane, allows the system to pivot left and right. This rotation changes the orientation of the attached device along the X-axis (horizontal axis) and the Z-axis (depth axis). This movement is used for aiming the camera horizontally across a plane parallel to the ground.

Rotation in the YZ Plane:
The servo represented in the top-right diagram is mounted perpendicularly to the first servo and allows the system to tilt up and down. This tilting motion adjusts the orientation of the camera along the Y-axis (vertical axis) and the Z-axis. This type of movement is used to change the elevation angle of the device, pointing it up or down.

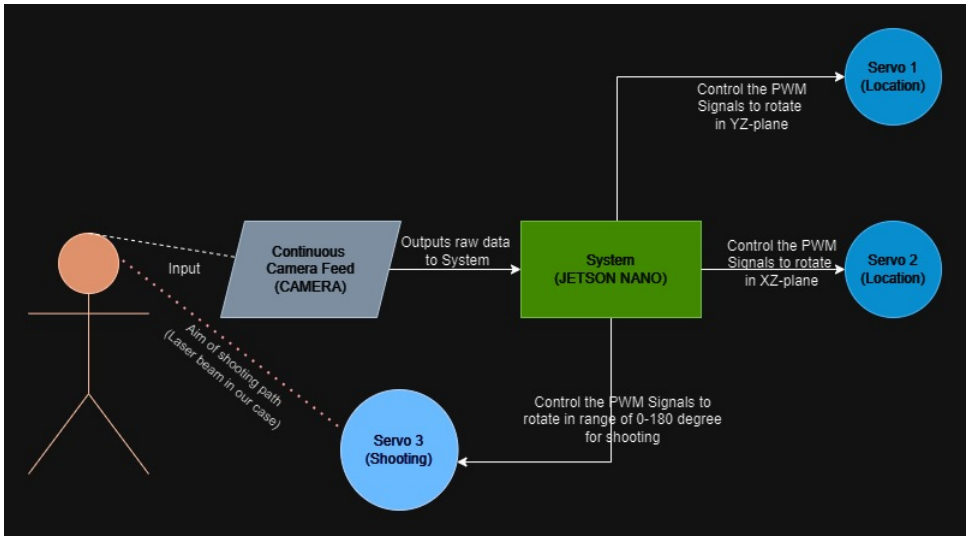

Figure 3: Working Overview

- Input - Continuous Camera Feed (CAMERA): The system begins with a camera that continuously captures video. This camera represents the primary sensor input for the system, continuously providing raw visual data.
- System (JETSON NANO): The raw camera feed is sent to the central processing unit, which is a Jetson Nano in this case. It processes the input data to detect faces and then calculates the necessary movements for the servo motors to align with the detected face.
- Servo 1 (Location): Based on the processed data from the Jetson Nano, Servo 1 adjusts the system's orientation in the YZ-plane (vertical plane). Control of the PWM (Pulse Width Modulation) signals to rotate in the YZ-plane suggests that Servo 1 can tilt the mechanism up or down to align vertically with the detected face.
- Servo 2 (Location): Simultaneously, Servo 2 receives control signals from the Jetson Nano to rotate in the XZ-plane (horizontal plane). This servo is responsible for the left-right movement, aligning the system horizontally with the target.
- Servo 3 (Shooting): Once the target is located and centered using Servos 1 and 2, Servo 3 is actuated. This servo controls the firing mechanism, and the diagram specifies that it can rotate in a range of 0-180 degrees to shoot, which implies a range of motion for the actuation mechanism, potentially to vary the firing angle or to engage a safety lock mechanism.
- The arrows indicate the direction of data and control flow. Starting from the camera input, moving to processing and decision-making on the Jetson Nano, and finally to the actuation of Servo 1 and Servo 2 for location alignment, and Servo 3 for shooting upon successful alignment.

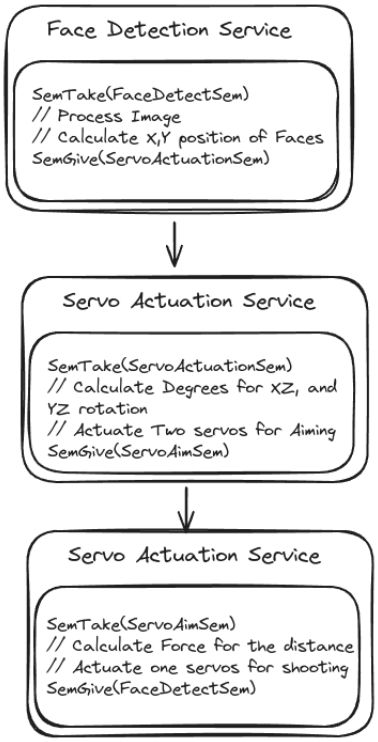

Figure 4: Dataflow Diagram :DFD

Face Detection Service: This service is responsible for detecting faces in images captured by the system.It receives input from a function called SemTake(FaceDetectSem), which suggests the presence of a synchronization mechanism (semaphore) to control access to the face detection service.The service processes the image, performs face detection, and calculates the X,Y position of the detected faces. After processing, it releases the semaphore using SemGive(ServoActuationSem), indicating that the face detection results are ready for the next service.

Servo Actuation Service: This service receives the face detection results from the Face Detection Service through the SemTake(ServoActuationSem) function. It calculates the necessary degrees for the X and Y axes to aim the servos towards the detected face. Based on the calculated degrees, it actuates two servos using the SemGive(ServoAimSem) function, which suggests passing control to the Servo Aiming Service.

Servo Aiming Service: This service takes control after receiving the signal from the Servo Actuation Service through SemTake(ServoAimSem).It calculates the force required to aim the servos based on the distance to the detected face.Finally, it actuates one servo for shooting, likely triggering a shooting mechanism.After the shooting action, it releases the control back to the Face Detection Service using SemGive(FaceDetectSem).

The arrows in the diagram represent the flow of control and data between the services. The use of semaphores (SemTake and SemGive) suggests a synchronization mechanism to ensure that each service operates in a coordinated manner and that data is passed correctly between them.

6 Question 6

Q: [10 points] Provide all major real-time service requirements with a description of each Si including Ci, Ti, and Di with a description of how the request frequency and deadline was determined for each as well as how Ci was determined, estimated or measured as WCET for the service. Deadlines must be real and related to a published standard; or physical requirement based on mathematical modeling; or research paper or study. Please provide this in your report for the final project as well as your Exercise 6 submission.

Answer: Service 1 (S1): Face Detection
Description: The face detection service captures video frames from the camera, detects faces in each frame using computer vision algorithms, and extracts the coordinates of the detected faces.

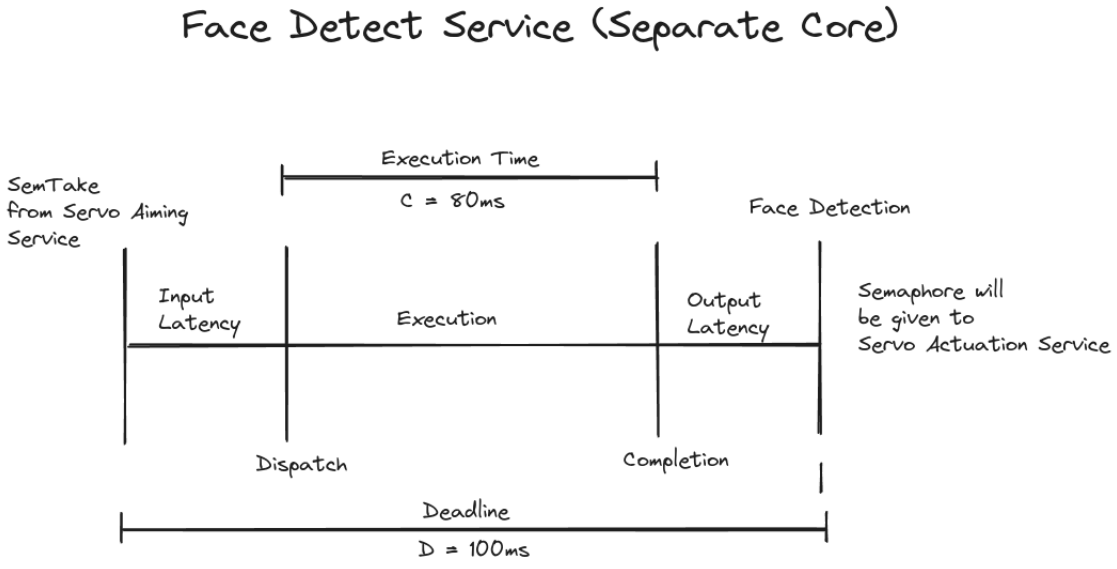

Figure 5: Face Detect Service Analysis

This service would run on different core than Servo Actuation and Servo Aiming Service

Execution Time (C1): 80 milliseconds
The period was chosen based on the requirement to process video frames at a rate of 10 frames per second (FPS) to achieve smooth and responsive face tracking. A period of 100 milliseconds ensures that the face detection service can process each frame within the allocated time.

Deadline (D1): 100 milliseconds
The deadline for the face detection service is set to be equal to its period (D1 = T1) to ensure that the processing of each frame is completed before the next frame arrives. This deadline is derived from the real-time requirement of processing video frames at 10 FPS for seamless face tracking.

Period (T1): 100 milliseconds
The period was chosen based on the requirement to process video frames at a rate of 10 frames per second (FPS) to achieve smooth and responsive face tracking. A period of 100 milliseconds ensures that the face detection service can process each frame within the allocated time.

Service 2 (S2): Servo Control

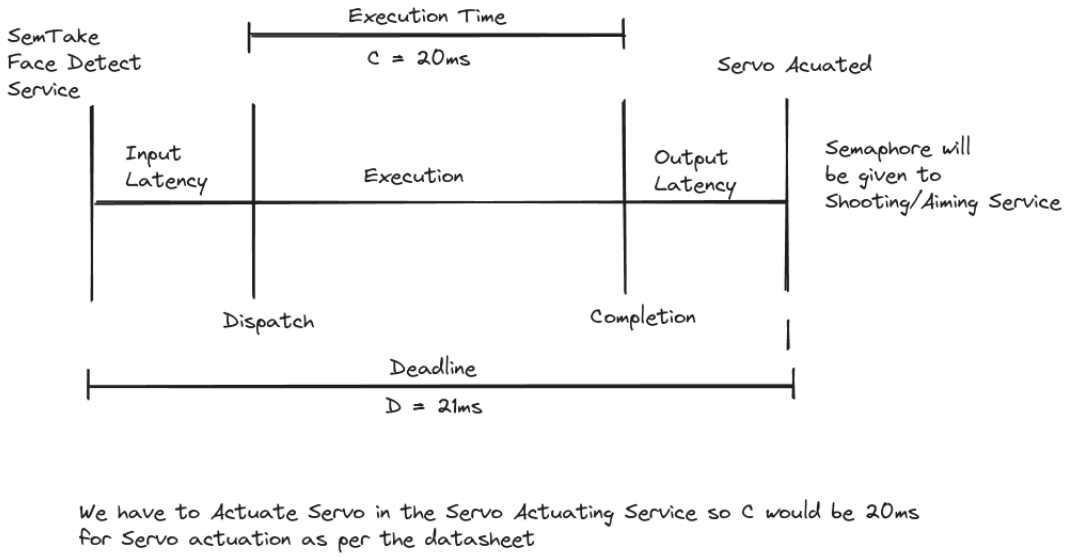

Figure 6: Servo Actuation Service Analysis

Description: The servo control service receives the face coordinates from the face detection service, calculates the required servo angles using trigonometric calculations, and sends control signals to the servos to aim at the detected face.

Execution Time (C2): 20 milliseconds The execution time for the servo control service was determined by measuring the WCET of the servo angle calculation and control signal generation on the Jetson Nano platform. The WCET was obtained by considering the maximum time required for the trigonometric calculations and servo communication.

Period (T2): 100 milliseconds The period for the servo control service is set to 21 milliseconds to ensure smooth and responsive servo movement. This period allows the servos to update their positions frequently enough to track the detected face accurately.

Deadline (D2): 21 milliseconds The deadline for the servo control service is to guarantee that the servo angles are calculated and control signals are sent within the allocated time. This deadline ensures that the servos can respond quickly to changes in the face position.

Service 3 (S3): Shooting Mechanism Control

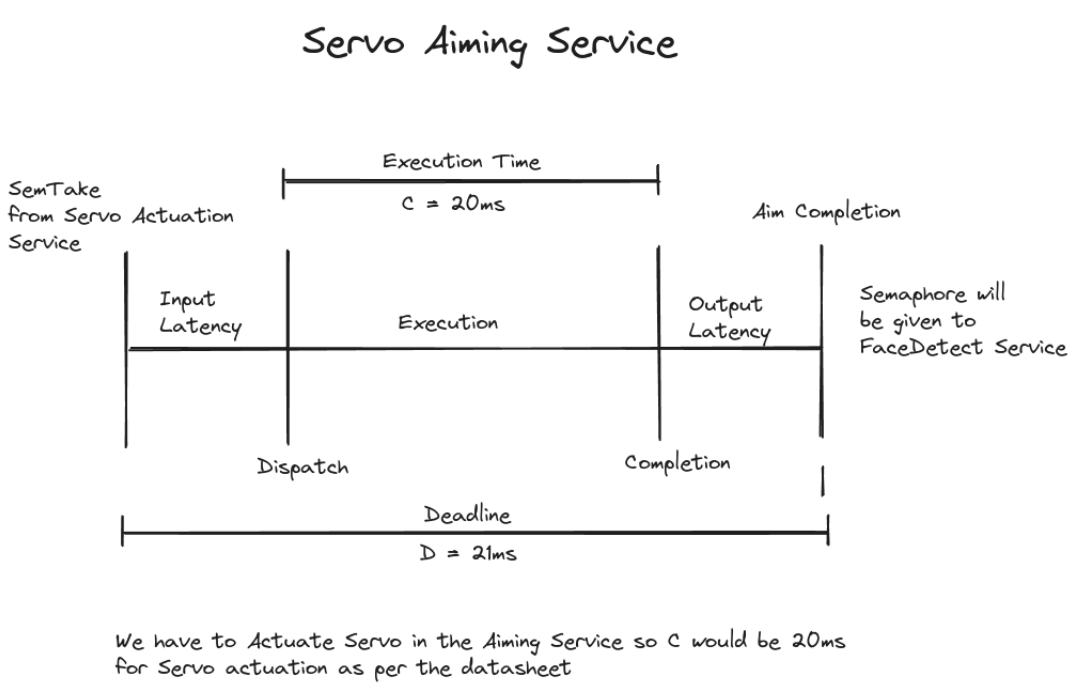

Figure 7: Servo Aiming Service Analysis

Description: The shooting mechanism control service monitors the alignment between the servos and the detected face, and triggers the shooting action when the alignment conditions are met.

Execution Time (C3): 20 milliseconds The execution time for the shooting mechanism control service was determined by measuring the WCET of the alignment check and trigger signal generation on the Jetson Nano platform. The WCET was obtained by considering the maximum time required for the alignment calculations and shooting mechanism communication.

Period (T3): 100 milliseconds The period for the shooting mechanism control service is set to 100 milliseconds, aligning with the face detection service's period. This period allows the shooting mechanism to be triggered in sync with the face detection and tracking process.

Deadline (D3): 21 milliseconds The deadline for the shooting mechanism control service is set to be equal to its period (D3 = T3) to ensure that the alignment check and trigger signal generation are completed within the allocated time. This deadline guarantees that the shooting mechanism can respond promptly to the detected face alignment.

Service	**Service name**	**C**	**WCET**	**D**	**T**
S1	Face Recognition Service	80ms	100ms(??)	100ms	100ms
S2	Servo Actuation Service	20ms	20ms(??)	21ms	100ms
S3	Servo Aiming Service	20ms	20ms(??)	21ms	100ms

Table 1: Estimated C,D,T

7 References

1. ECEN 5623 Lecture slides material and example codes.
2. REAL-TIME EMBEDDED COMPONENTS AND SYSTEMS with LINUX and RTOS, Sam Siewert John Pratt (Chapter 10, 11 & 12).
3. Exercise 6 requirements included links and documentation.

Bibliography

Amir Benzaoui Insaf Adjabi, Abdeldjalil Ouahabi and Abdelmalik Taleb-Ahmed. Past, present, and future of face recognition: A review. https://www.mdpi.com/2079-9292/9/8/1188, 2020. Accessed: 2020-06-19.

OpenCV. Opencv face detection. https://docs.opencv.org/3.4/da/d60/tutorial_face_main.html, 2023. Accessed: 2023-06-19.

Thair Salih and Mohammad Gh. A novel face recognition system based on jetson nano developer kit. *IOP Conference Series: Materials Science and Engineering*, 928:032051, 11 2020. doi: 10.1088/1757-899X/928/3/032051.

Vishwani Sati, Sergio Márquez-Sánchez, Niloufar Shoeibi, Ashish Arora, and Juan Corchado Rodríguez. *Face Detection and Recognition, Face Emotion Recognition Through NVIDIA Jetson Nano*, pages 177–185. 09 2020. ISBN 978-3-030-58355-2. doi: 10.1007/978-3-030-58356-9_18.

Appendices

A C Code for the Implementation

facedetect.cpp

```cpp
// CPP program to detects face in a video

// Include required header files from OpenCV directory
#include "opencv2/objdetect.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include <iostream>

using namespace std;
using namespace cv;

// Function for Face Detection
void detectAndDraw( Mat& img, CascadeClassifier& cascade,
                CascadeClassifier& nestedCascade, double scale );
string cascadeName, nestedCascadeName;

int main( int argc, const char** argv )
{
    // VideoCapture class for playing video for which faces to be detected
    VideoCapture capture;
    Mat frame, image;

    // PreDefined trained XML classifiers with facial features
    CascadeClassifier cascade, nestedCascade;
    double scale=1;

    // Load classifiers from "opencv/data/haarcascades" directory
    nestedCascade.load( "./haarcascade_eye_tree_eyeglasses.xml" ) ;

    // Change path before execution
    cascade.load( "./haarcascade_frontalcatface.xml" ) ;

    // Start Video..1) 0 for WebCam 2) "Path to Video" for a Local Video
    capture.open(0);
    if( capture.isOpened() )
    {
        // Capture frames from video and detect faces
        cout << "Face Detection Started...." << endl;
        while(1)
        {
            capture >> frame;
            if( frame.empty() )
                break;
            Mat frame1 = frame.clone();
            detectAndDraw( frame1, cascade, nestedCascade, scale );
            char c = (char)waitKey(10);

            // Press q to exit from window
            if( c == 27 || c == 'q' || c == 'Q' )
                break;
        }
    }
    else
```

```
54          cout<<"Could not Open Camera";
55      return 0;
56  }
57
58  void detectAndDraw( Mat& img, CascadeClassifier& cascade,
59                      CascadeClassifier& nestedCascade,
60                      double scale)
61  {
62      vector<Rect> faces, faces2;
63      Mat gray, smallImg;
64
65      cvtColor( img, gray, COLOR_BGR2GRAY ); // Convert to Gray Scale
66      double fx = 1 / scale;
67
68      // Resize the Grayscale Image
69      resize( gray, smallImg, Size(), fx, fx, INTER_LINEAR );
70      equalizeHist( smallImg, smallImg );
71
72      // Detect faces of different sizes using cascade classifier
73      cascade.detectMultiScale( smallImg, faces, 1.1,  2, 0|CASCADE_SCALE_IMAGE,
    Size(30, 30) );
74
75      // Draw circles around the faces
76      for ( size_t i = 0; i < faces.size(); i++ )
77      {
78          Rect r = faces[i];
79          Mat smallImgROI;
80          vector<Rect> nestedObjects;
81          Point center;
82          Scalar color = Scalar(255, 0, 0); // Color for Drawing tool
83          int radius;
84
85          double aspect_ratio = (double)r.width/r.height;
86          if( 0.75 < aspect_ratio && aspect_ratio < 1.3 )
87          {
88              center.x = cvRound((r.x + r.width*0.5)*scale);
89              center.y = cvRound((r.y + r.height*0.5)*scale);
90              radius = cvRound((r.width + r.height)*0.25*scale);
91              circle( img, center, radius, color, 3, 8, 0 );
92          }
93          else
94              // rectangle( img, cvPoint(cvRound(r.x*scale), cvRound(r.y*scale)),
    cvPoint(cvRound((r.x + r.width-1)*scale), cvRound((r.y + r.height-1)*scale)), color,
    3, 8, 0);
95          if( nestedCascade.empty() )
96              continue;
97          smallImgROI = smallImg( r );
98
99          // Detection of eyes in the input image
100         nestedCascade.detectMultiScale( smallImgROI, nestedObjects, 1.1, 2,
    0|CASCADE_SCALE_IMAGE, Size(30, 30) );
101
102         // Draw circles around eyes
103         for ( size_t j = 0; j < nestedObjects.size(); j++ )
104         {
105             Rect nr = nestedObjects[j];
106             center.x = cvRound((r.x + nr.x + nr.width*0.5)*scale);
```

```
107             center.y = cvRound((r.y + nr.y + nr.height*0.5)*scale);
108             radius = cvRound((nr.width + nr.height)*0.25*scale);
109             circle( img, center, radius, color, 3, 8, 0 );
110         }
111     }
112
113     // Show Processed Image with detected faces
114     imshow( "Face Detection", img );
115 }
116
```