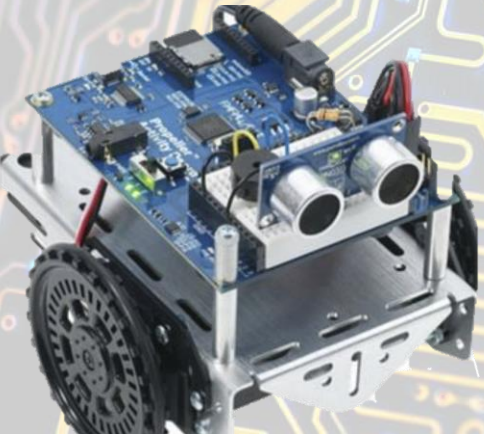
EC535 Introduction to Embedded Systems



EC535 2025 Final Project Grading Criteria

Item	Grading	Score	Comments
Demo/ presentation	Successful project completion, performance of the demo, consideration of system as a whole, video.	20	What was the level of technical accomplishment for this project, the presentation sufficiently shows the project.
Technical Report	Written report to be submitted on 5/1.	70	Clarity, organization, and completeness of figures, diagrams, see below.
Git/commits	Completeness and adherence to weekly commit schedule	10	Commits by both team members, starting to work early (two weeks prior to presentation).

Total: 100

The goal of the final project is to show understanding of concepts from the class and implement into an integrated system that consider functionality – some task needs to be achieved, e.g., autonomous vehicle responses on time and arrives to a goal, user interface enables task completion, etc. The project should draw on concepts in embedded and resource-constrained systems. Project examples can be found here:

https://docs.google.com/document/d/1oCxSCoXTv5vd8RuECxYzjg5Jnyiyxf6Fi64txKUYcOM/edit?usp=sharing

Example report can be found here:

https://drive.google.com/drive/folders/1D5XTPkJQXZXrxm9T4qOLDZvu1a7rzL6L?usp=sharing

Overall / Video (5 minutes):

Please produce a video showcasing your project. In this video, you should show the totality of your project:

- Provide an executive overview of your project – what it is designed to do, why, etc.
- Walk through building and deploying your project (30 seconds)
- Talk in great detail about your technology stack, and show that you understand why you are using the tech. For instance, "I push this button, it is captured by the kernel module that I wrote, which then calls the userland application to do ..."
- The video doesn't have to be polished, but it should have great detail about the technical stack and the technical work that went into the project: project overview, method, results and examples, and conclusion and future work.
- Please put the video in your github repository and upload it to YouTube as an unlisted (not private) video.
- In class presentation logistics and details will be released closer to the deadline.

Technical Report (minimum 4 pages):

The report should contain:

An abstract: a paragraph with describes the research problem, and an overview of the planned project (5 pts).

An introduction: well-formed motivation, background information (10 pts).

Method: Method used is clearly explained, with your contributions, if you used ideas from our class readings or other materials, make sure to detail in the report. Your report should still be self-contained by including the used equations and formulations necessary for understanding the approach, any algorithms. The method should be well justified, and consider complete system usability in the real-world (20 pts).

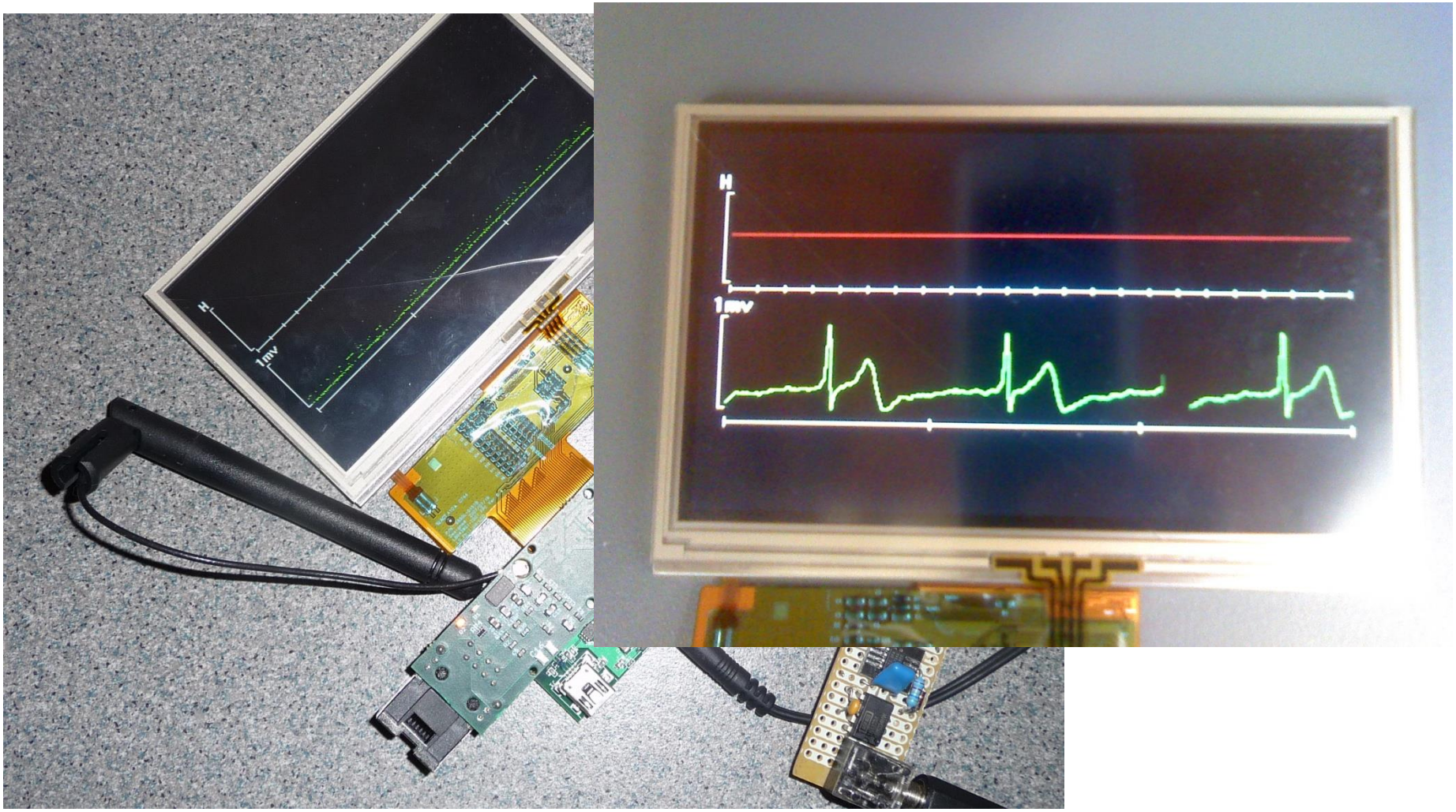
Results: Experimental setup should be clearly explained. Was any data used, evaluation protocol and metric definition details, are the experiments well justified, analyze the system comprehensively (with quantitative and qualitative results) (30 pts).

Limitations and future work: Discuss what didn't work, e.g., if different from expectations, what wanted to implement but didn't have time (e.g., to further tune), conclusion (5 pts).

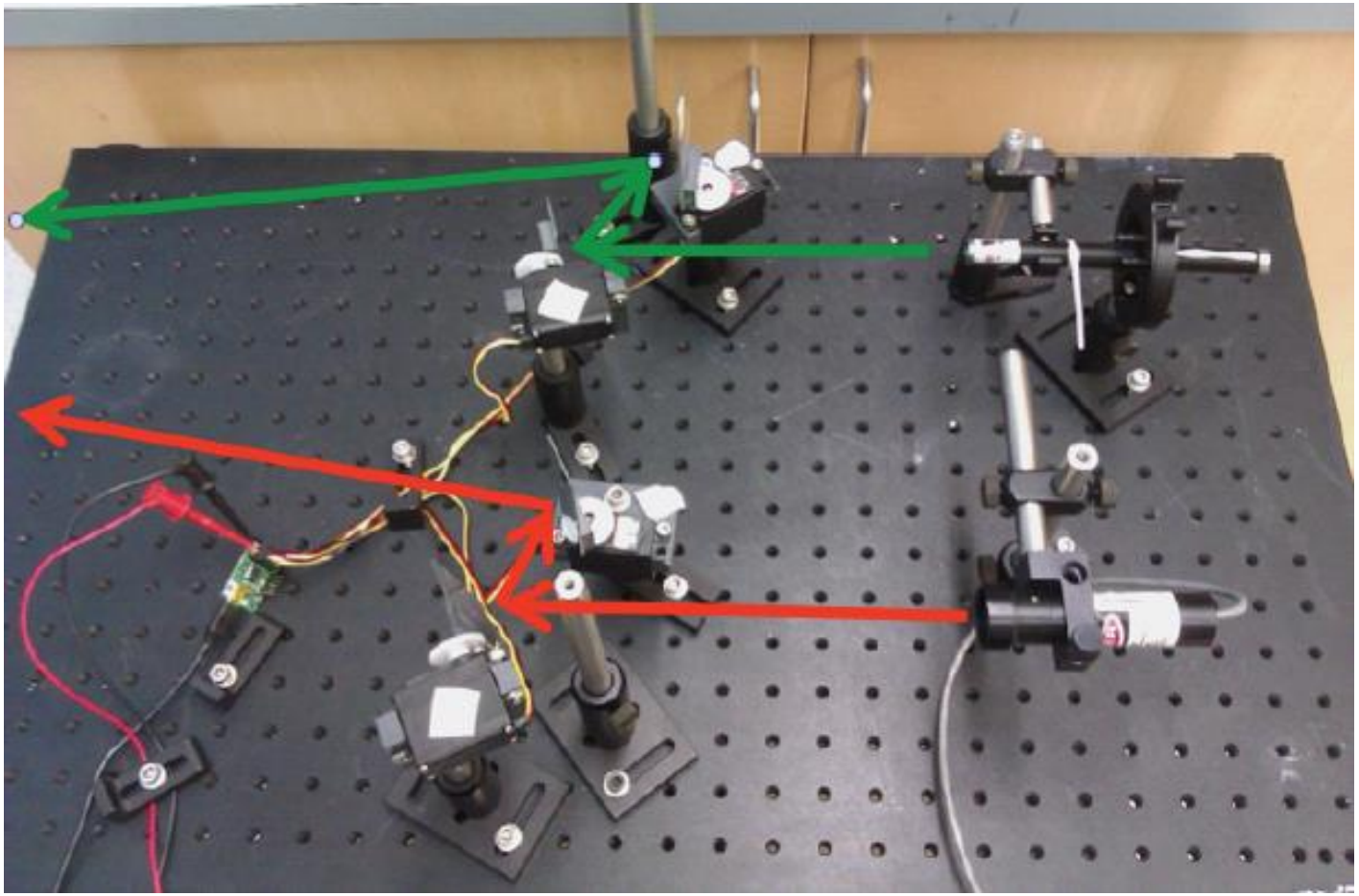
Additional items to consider:

- Aspects of the project which were not implemented and why (we ran out of time is not a good reason for why a project doesn't work). Focus on specific reasons.
- Aspects which were implemented but not in original proposal, maybe you did something beyond the scope of your original project, or you had to change a component that was in the original project.
- Show a software/schematic/network architecture of how things are interconnected
- References to papers, websites, etc.
- What (technical skills/knowledge) did you learn from your project?

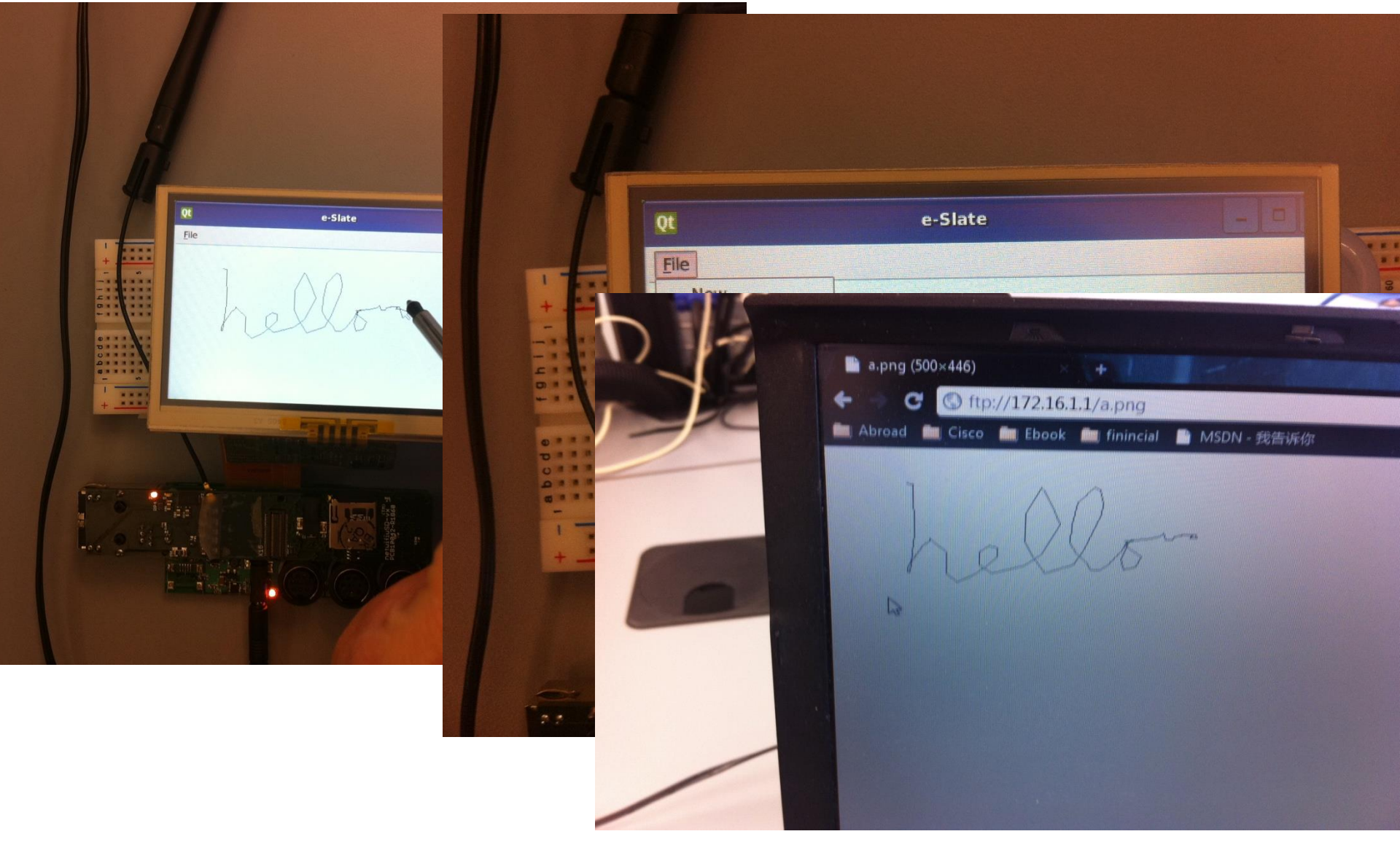
Wireless heart monitor



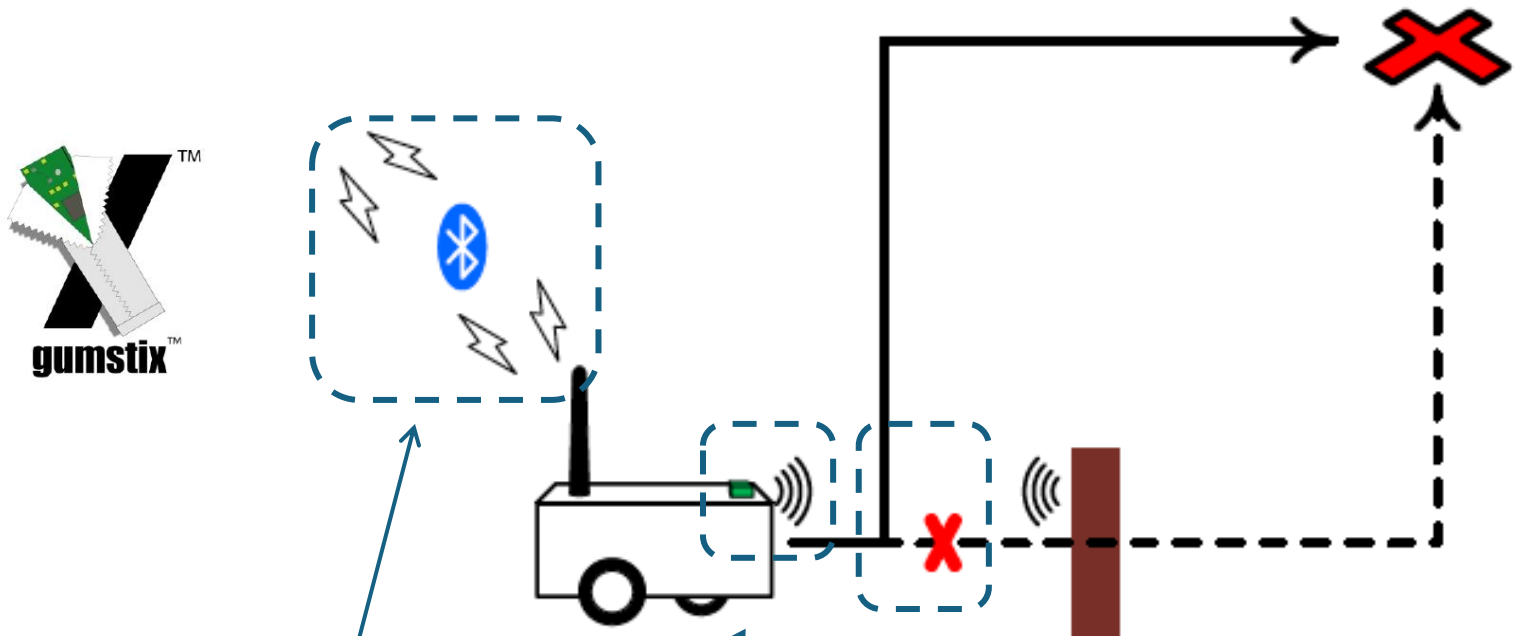
Multi-screen laser pointer using servo control



e-Slate: mobile handwriting notepad

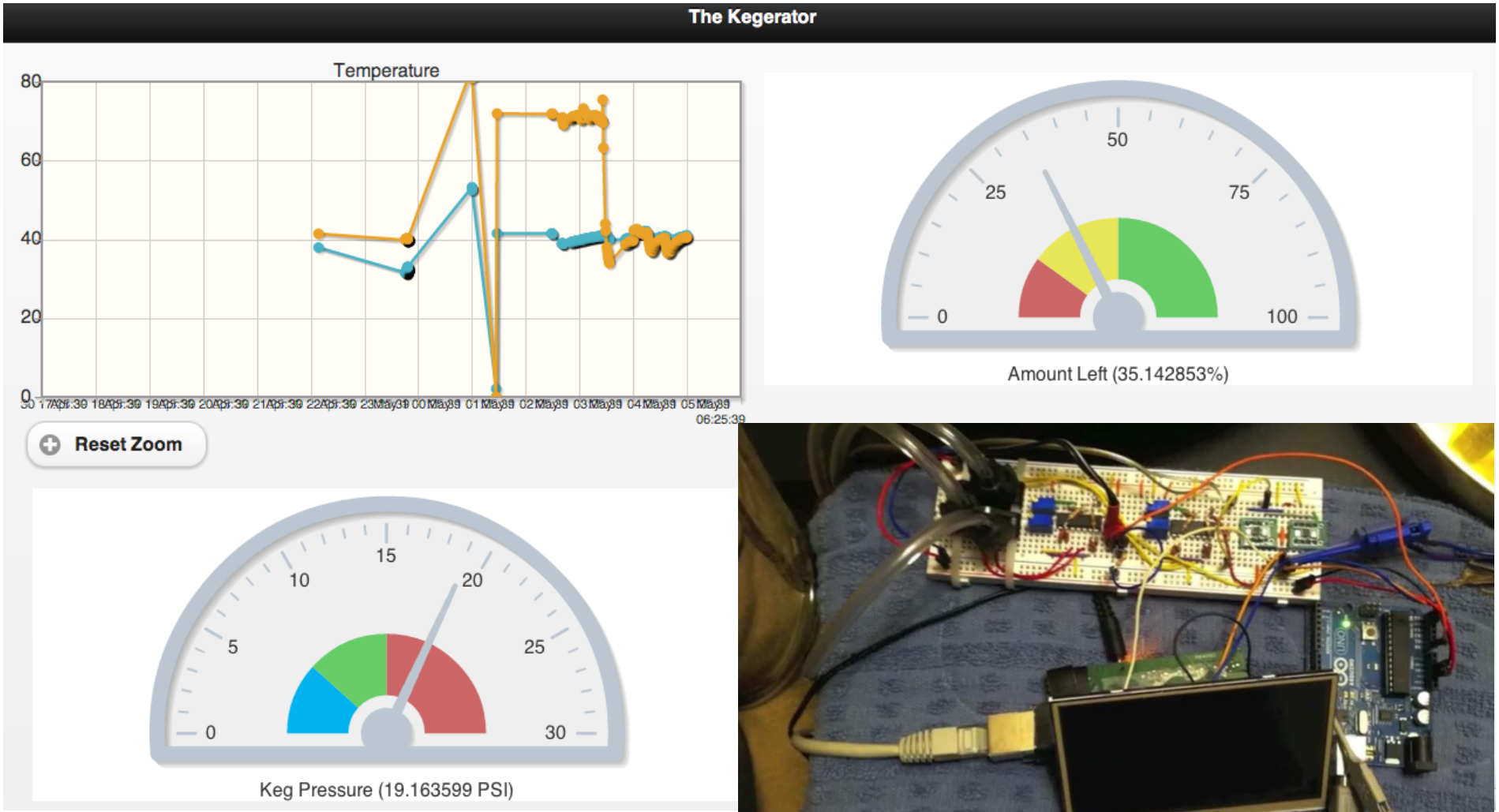


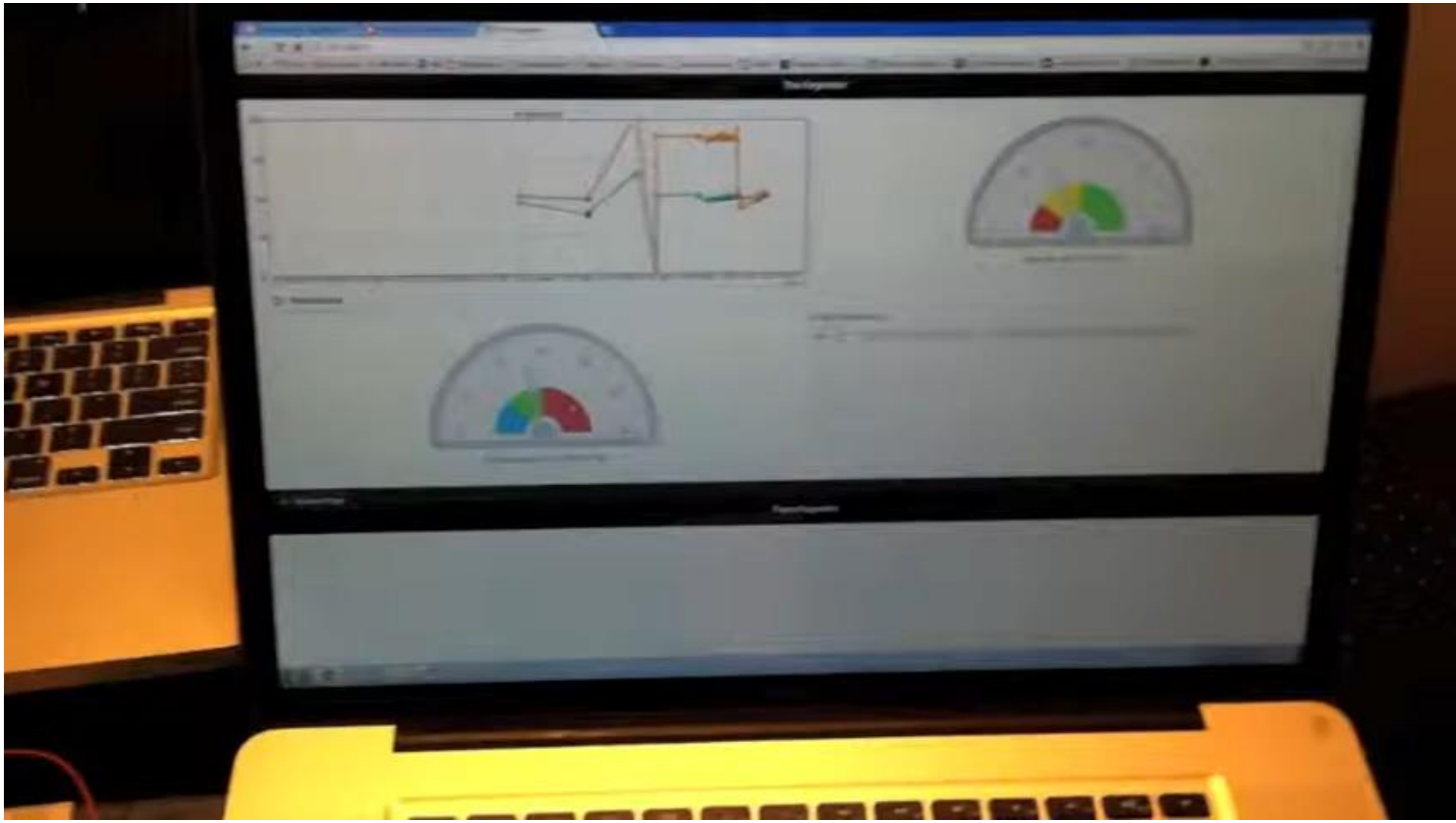
Bluetooth-controlled vehicle



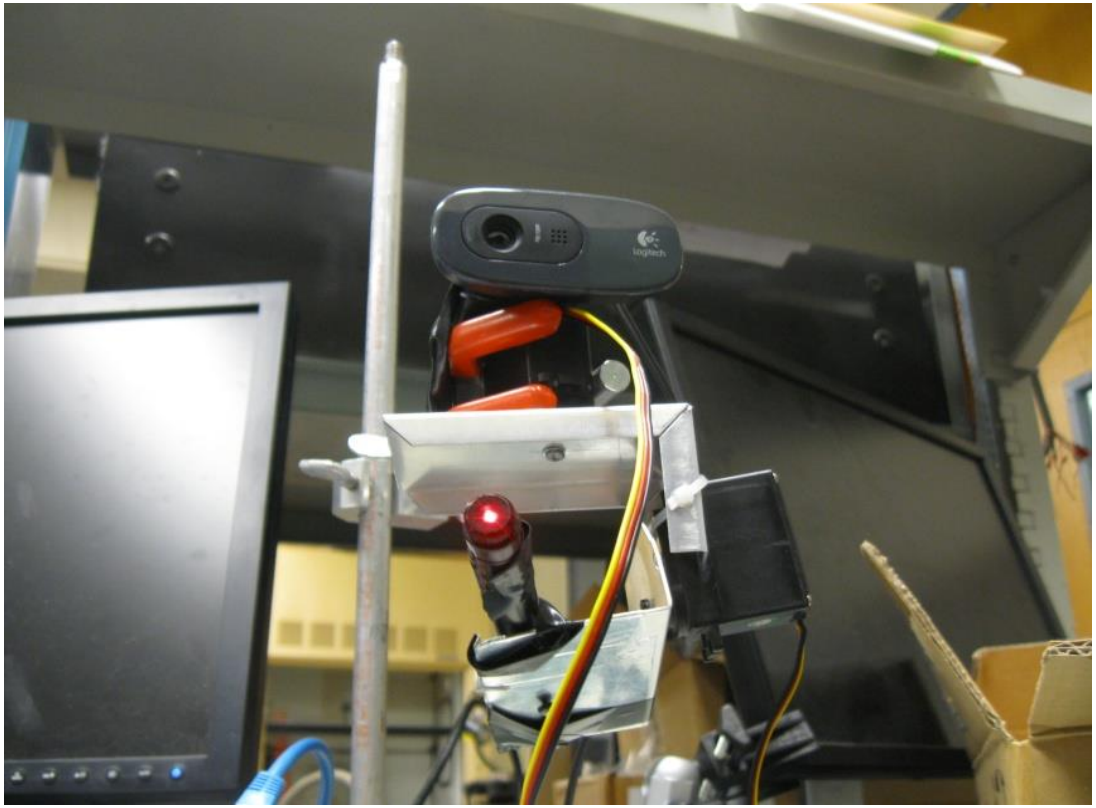
- **Bluetooth Communication – Assign the destination**
  - **Infrared Sensor – Distance between the car and the obstacle**
    - **Self Routing – Take another path to detour the obstacle**

Home-brew monitoring & actuating system (Kegerator)

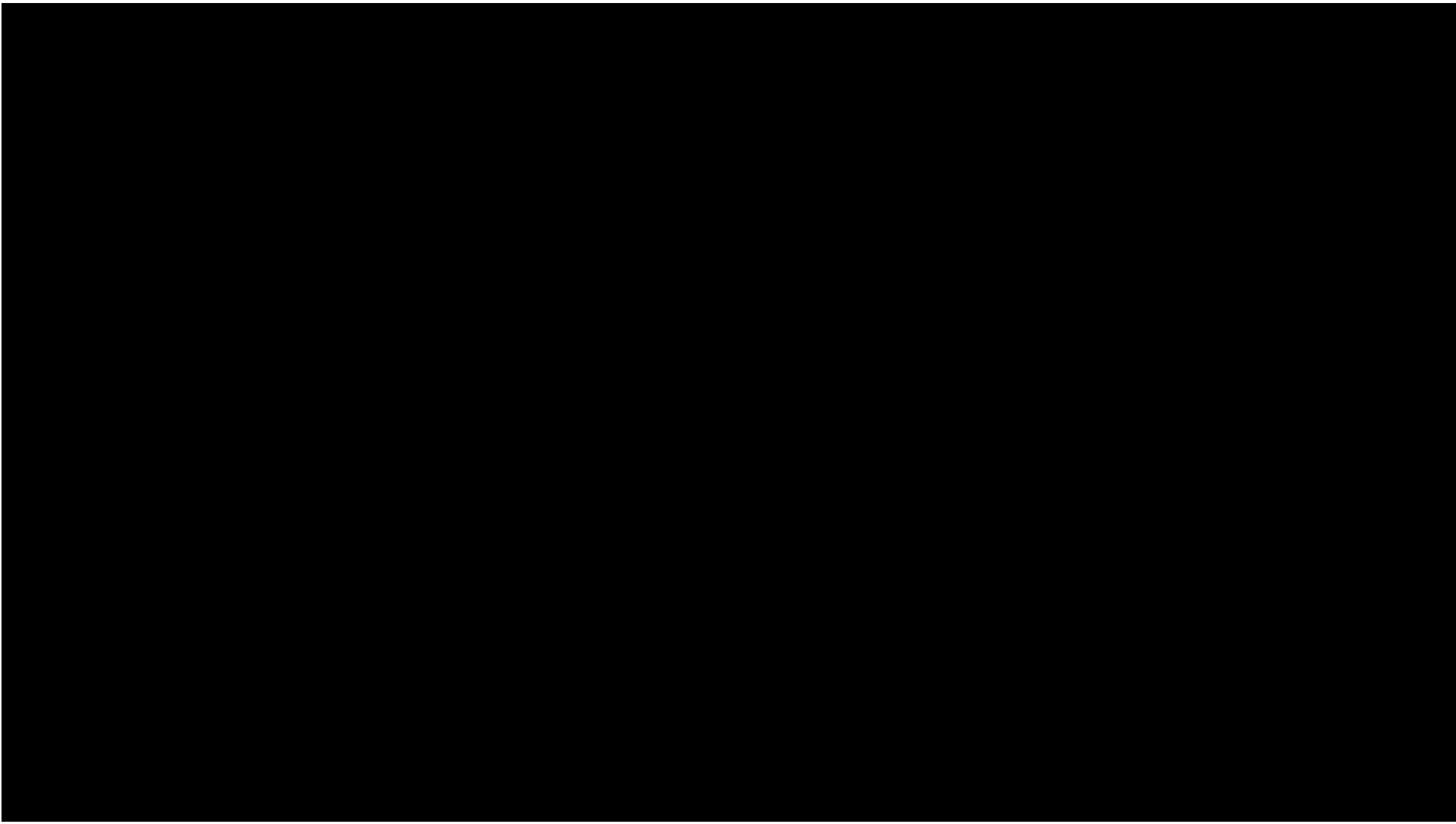




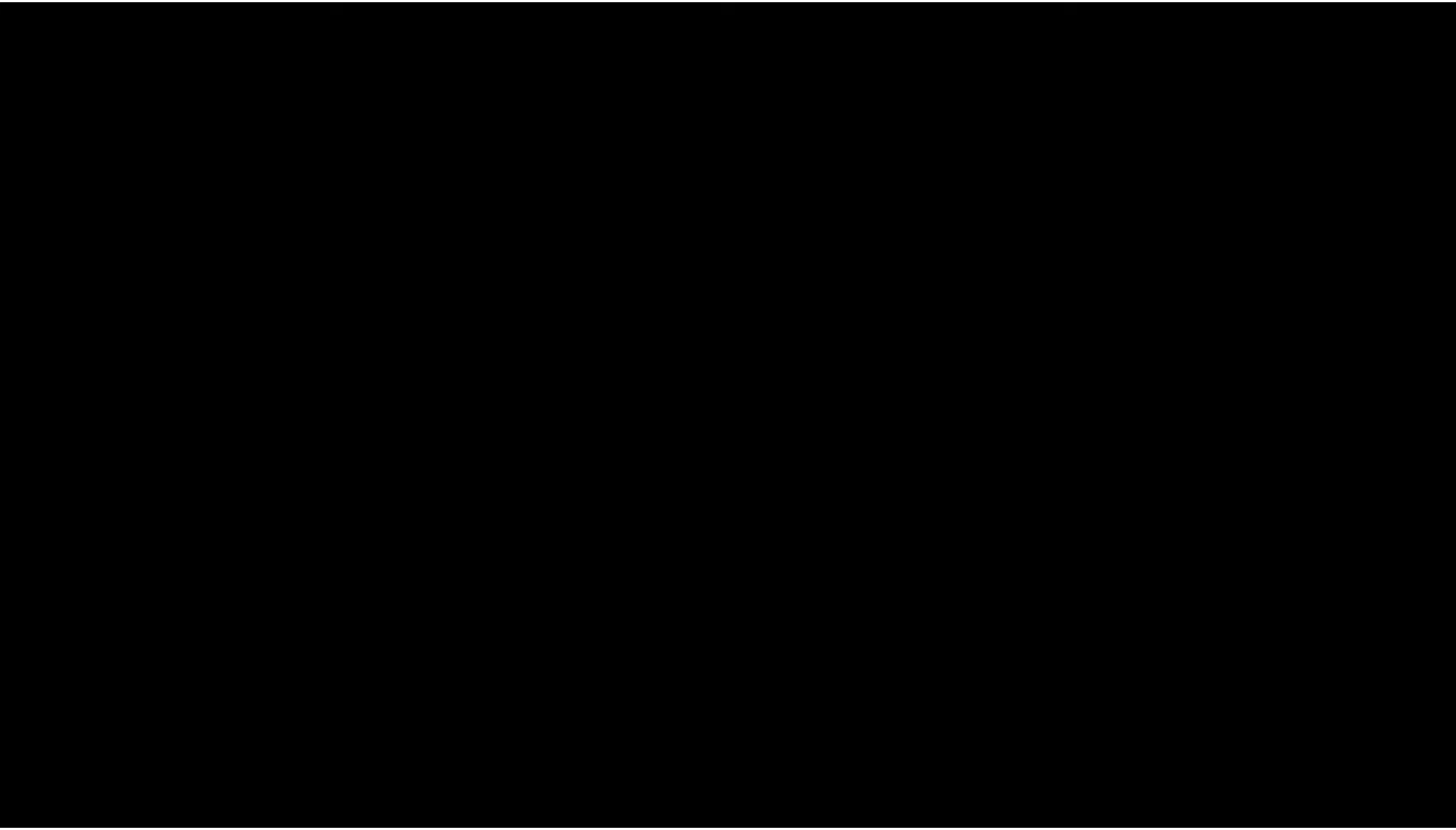
Laser Turret



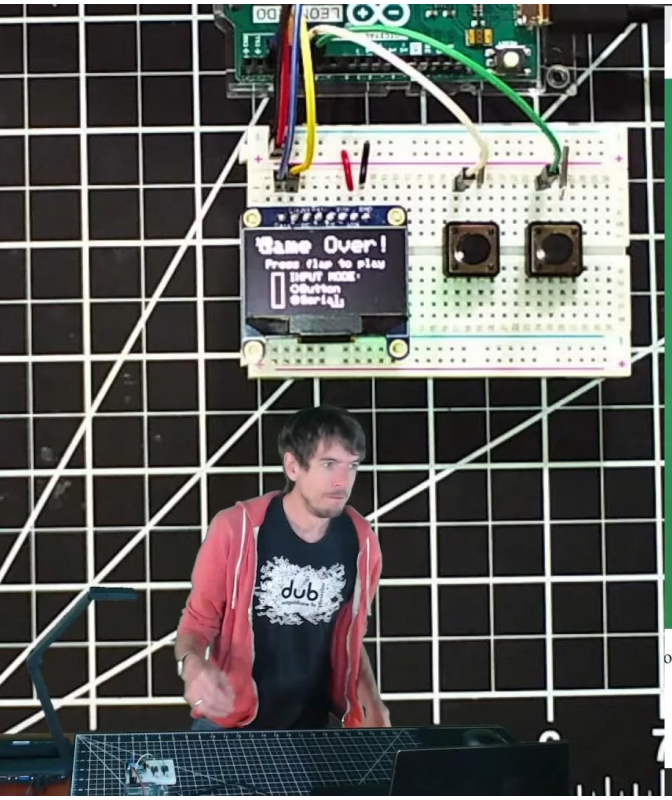
Smartwatch controlled car

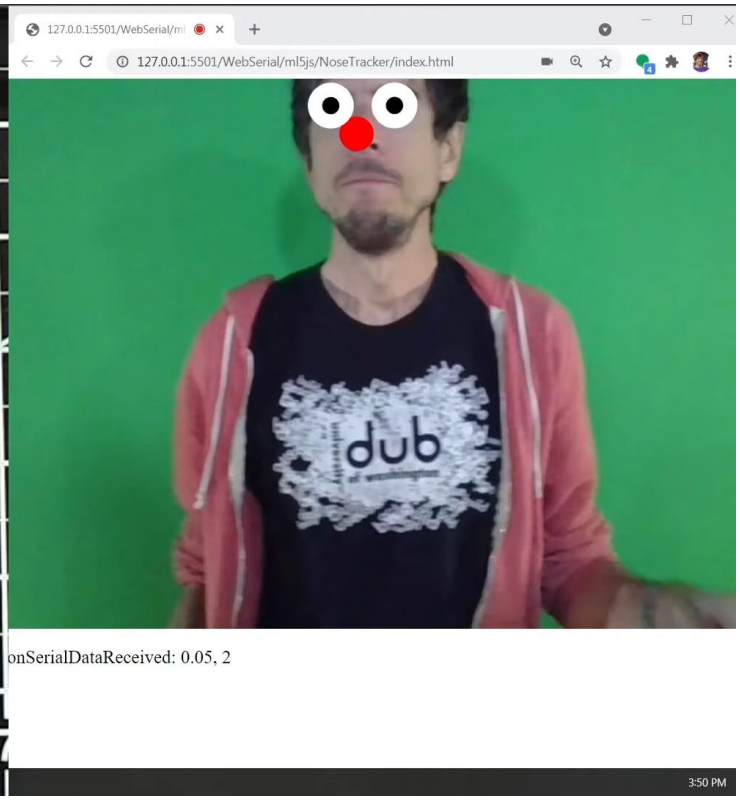


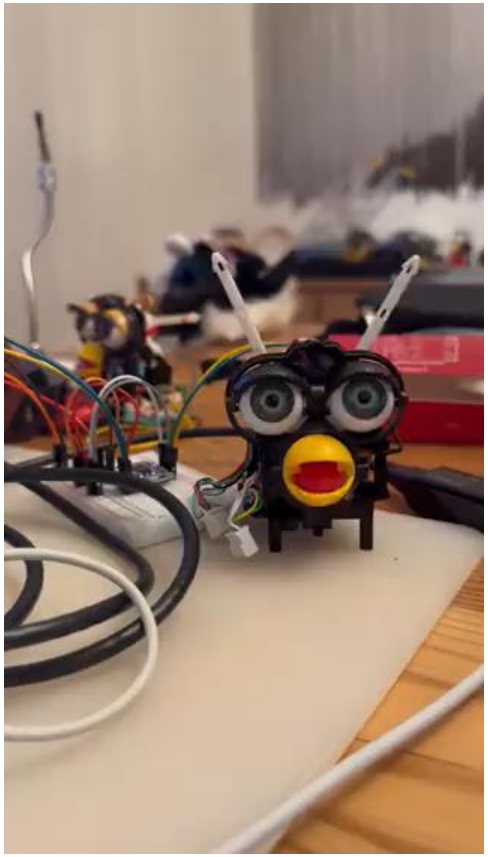
EMG Controlled Robotic Arm (we have one)



Beach Brawl


Game Over!
Press fire to play


127.0.0.1:5501/WebSerial/ml
127.0.0.1:5501/WebSerial/ml5js/NoseTracker/index.html
dub
onSerialDataReceived: 0.05, 2
3:50 PM



- Multimodal LLMs on Smartphone

Incorporate efficiency constraints for mobile LLMs or vision-based LLMs to implement on a smartphone, prompt to perform a task, e.g., assistive navigation.

- Scheduling/Efficient Policies for Autonomous Vehicles

Implement on a real-world platform various scheduling policies to ensure timeliness and responsiveness to various events.

Can be based on our ECCV paper
https://github.com/unilcd/unilcd.github.io/blob/main/resources/UniLCD_paper.pdf

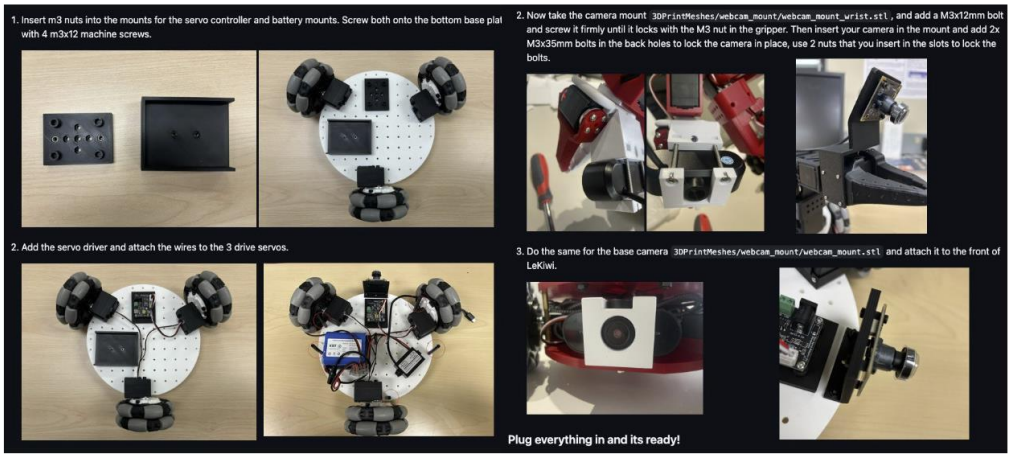
- Tiny Robotic Policies

Implement and develop your own operational microvehicle based on:

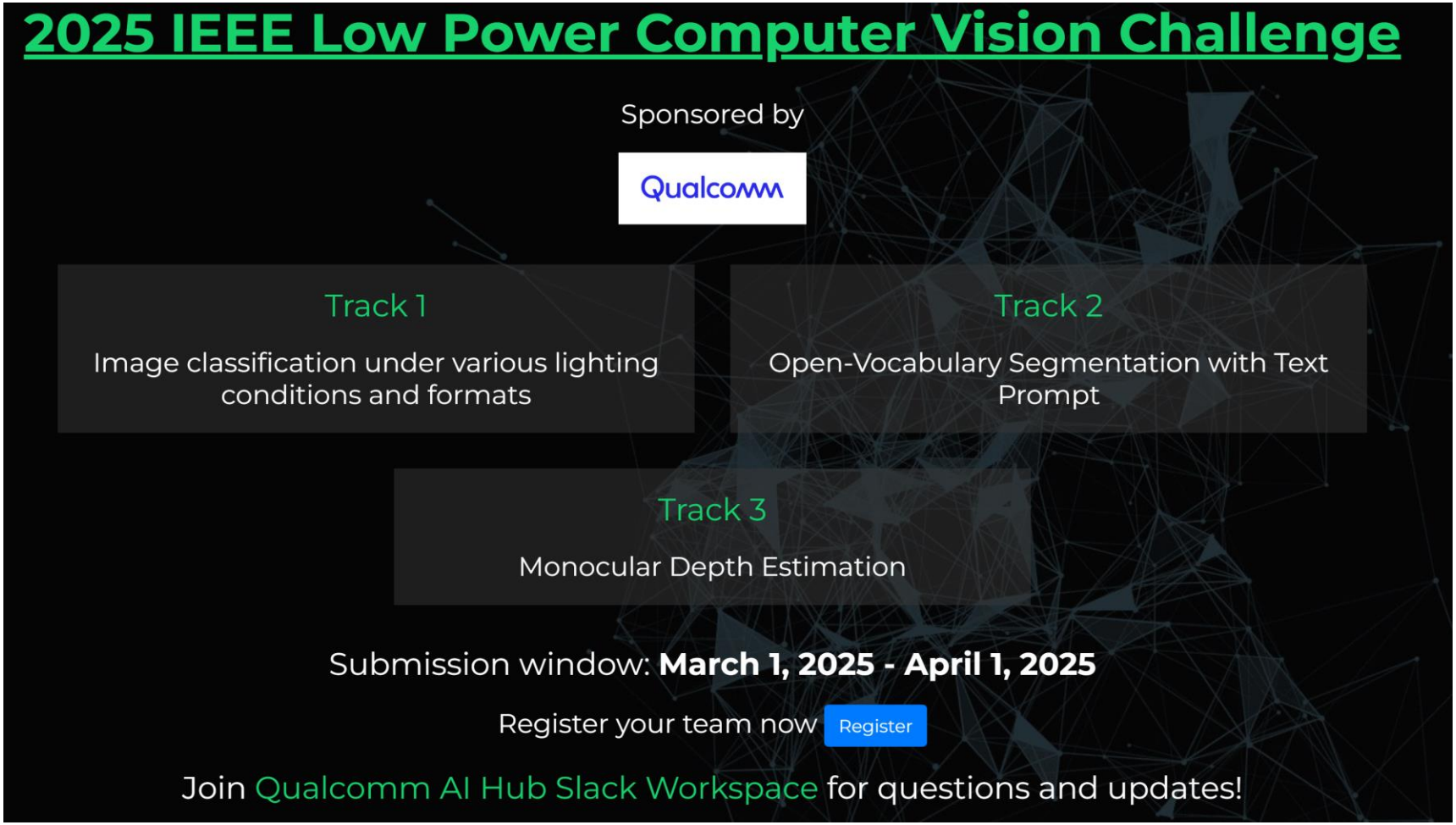
https://arxiv.org/pdf/2502.12355

https://dspace.mit.edu/handle/1721.1/139867

Also see:
https://bsky.app/profile/remicadene.bsky.social/post/3ljhwlyq4yl2i
https://github.com/huggingface/lerobot



2025 IEEE LOW-POWER COMPUTER VISION CHALLENGE



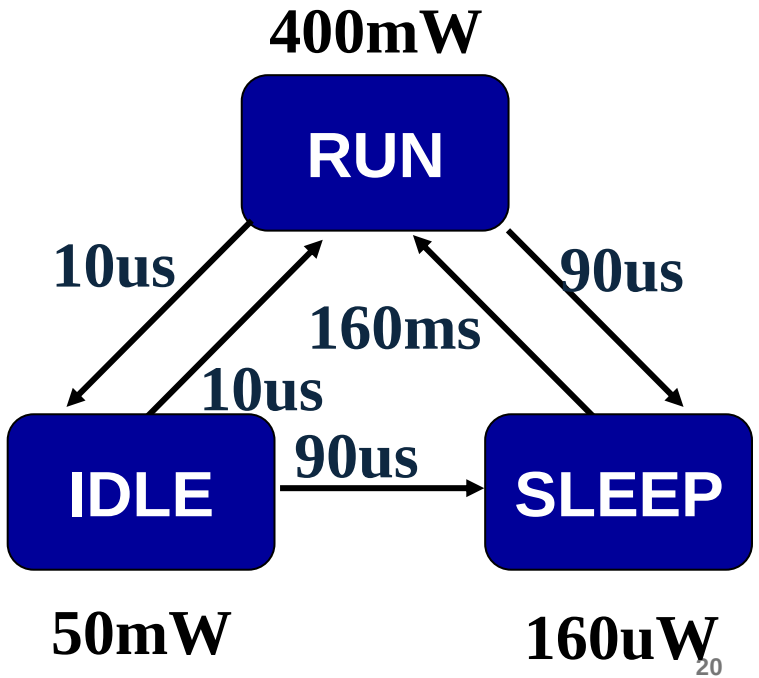
https://lpcv.ai/

Power Manageable Components

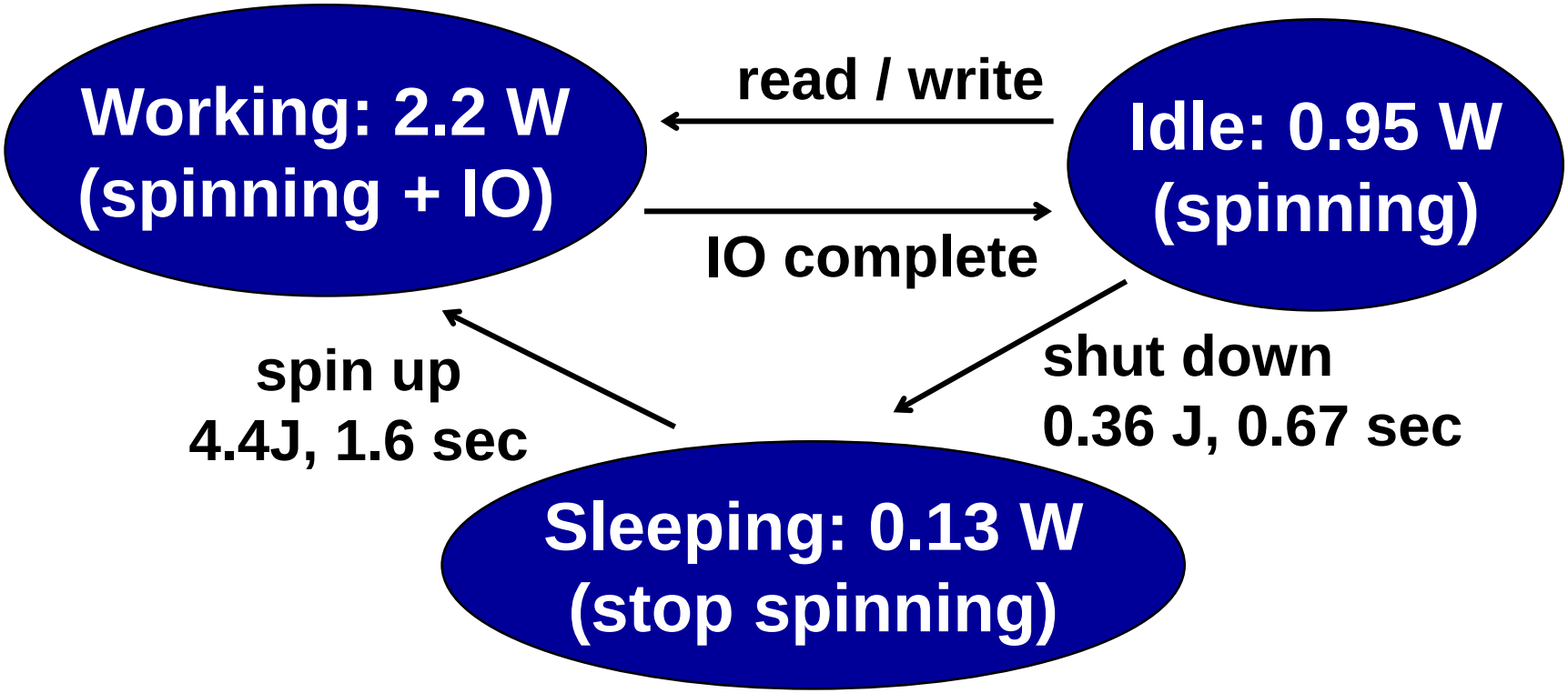
- Components with several internal states
  - Corresponding to power and service levels
- Abstracted as **power state machines**
  - State diagram with:
    - Power and service annotation on states
    - Power and delay annotation on edges

Example: SA-1100

- **RUN: Operational**
- **IDLE: A SW routine may stop the CPU when not in use, while monitoring interrupts**
- **SLEEP: Shutdown of on-chip activity**

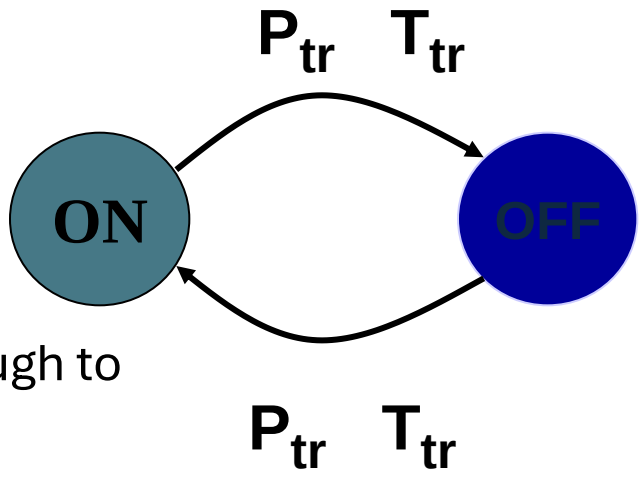


Example: Hard Disk Drive

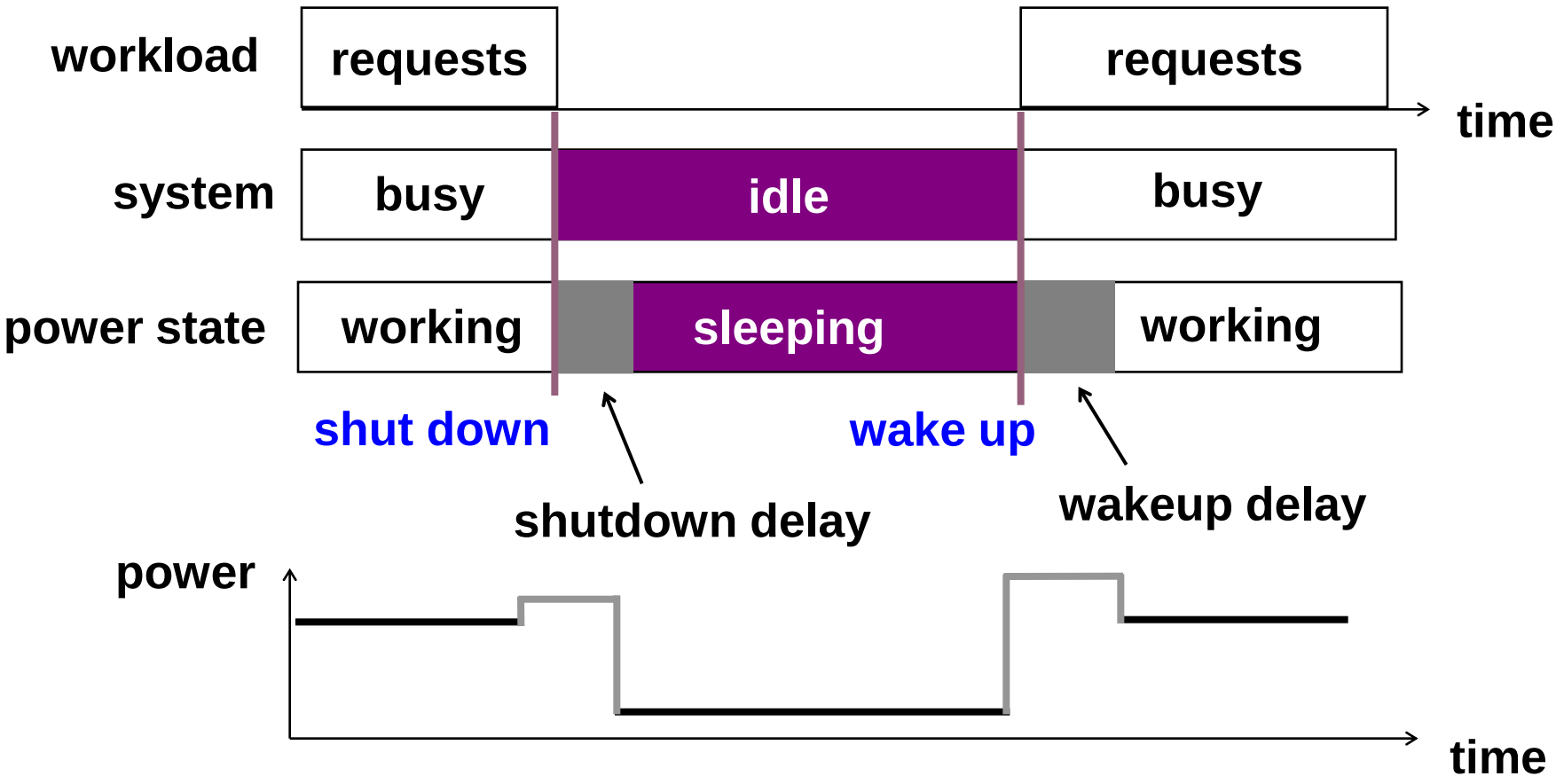


Dynamic Power Management

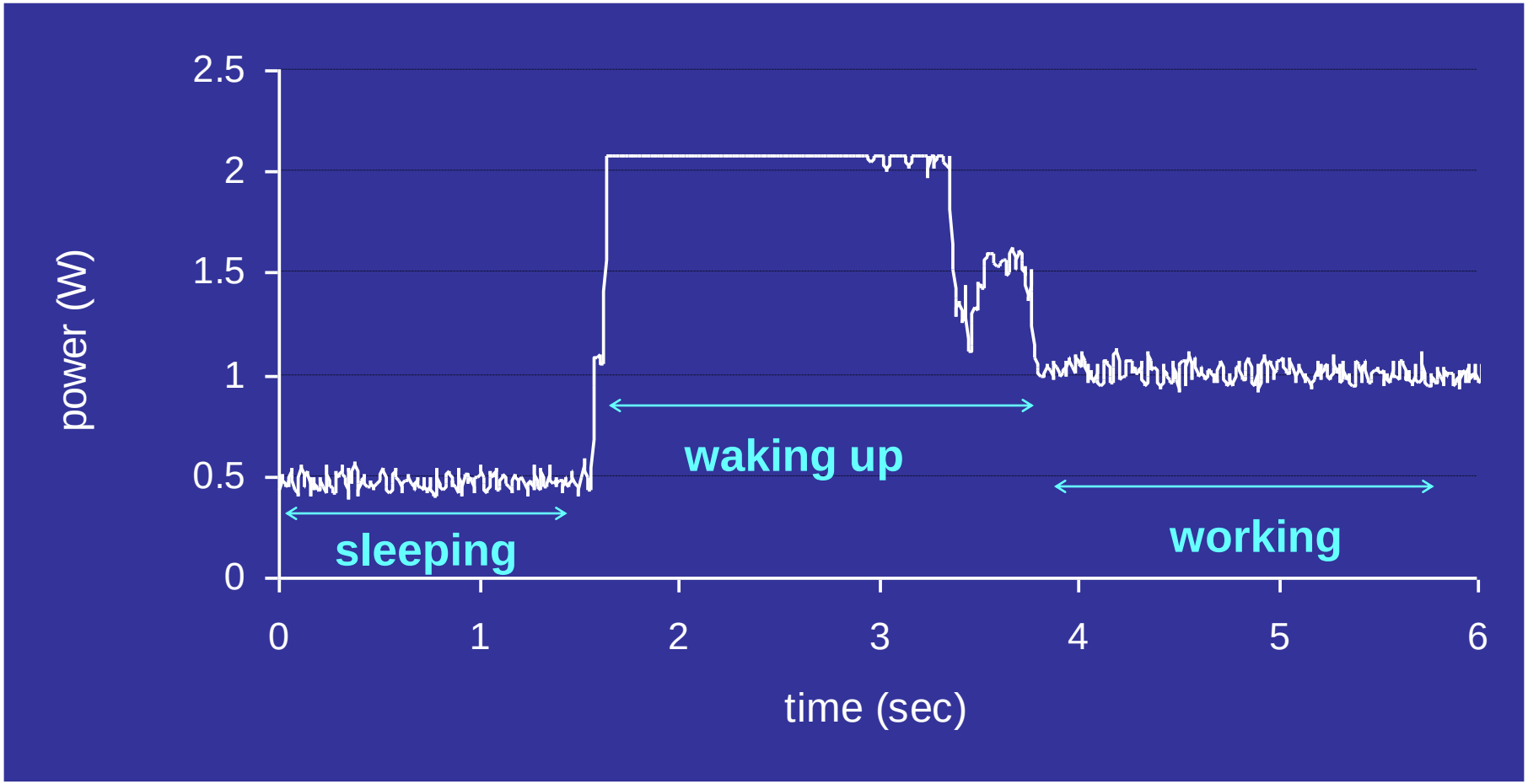
- State transition power (P_{tr}) and delay (T_{tr})
- If $T_{tr} = 0$, $P_{tr} = 0$ the policy is trivial
  - Stop a component when it is not needed
- If $T_{tr} != 0$ or $P_{tr} != 0$ (always…)
  - E.g., XScale 27x
    - 0.5 ms to sleep state
  - Shutdown only when idleness is long enough to neglect the cost
  - What if T and P fluctuate?



Workload and System Representation



Waking Up Hard Disk



Measurements done on a Fujitsu hard disk

Car Example

Imagine a car engine:
•If you turn it off for 5 seconds but it takes 5 seconds' worth of fuel to restart, you've wasted energy instead of saving any.

•If you leave it off for longer than 5 seconds, then you start seeing energy savings.

The system needs to be OFF long enough to recover the transition energy loss.

Car Example

Imagine a car engine:
•If you turn it off for 5 seconds but it takes 5 seconds' worth of fuel to restart, you've wasted energy instead of saving any.

•If you leave it off for longer than 5 seconds, then you start seeing energy savings.

The system needs to be OFF long enough to recover the transition energy loss.

-> System Break-Even Time

System Break-Even Time: *TBE*

Minimum idle time for amortizing the cost of component shutdown

$$T_{BE} = T_{tr} + T_{tr} \frac{P_{tr} - P_{on}}{P_{on} - P_{off}}$$

Transition delay (T_{tr})

Transition power (P_{tr})

Sleep power (P_{off})

Decision-Making

If $T_{idle} < T_{BE}$

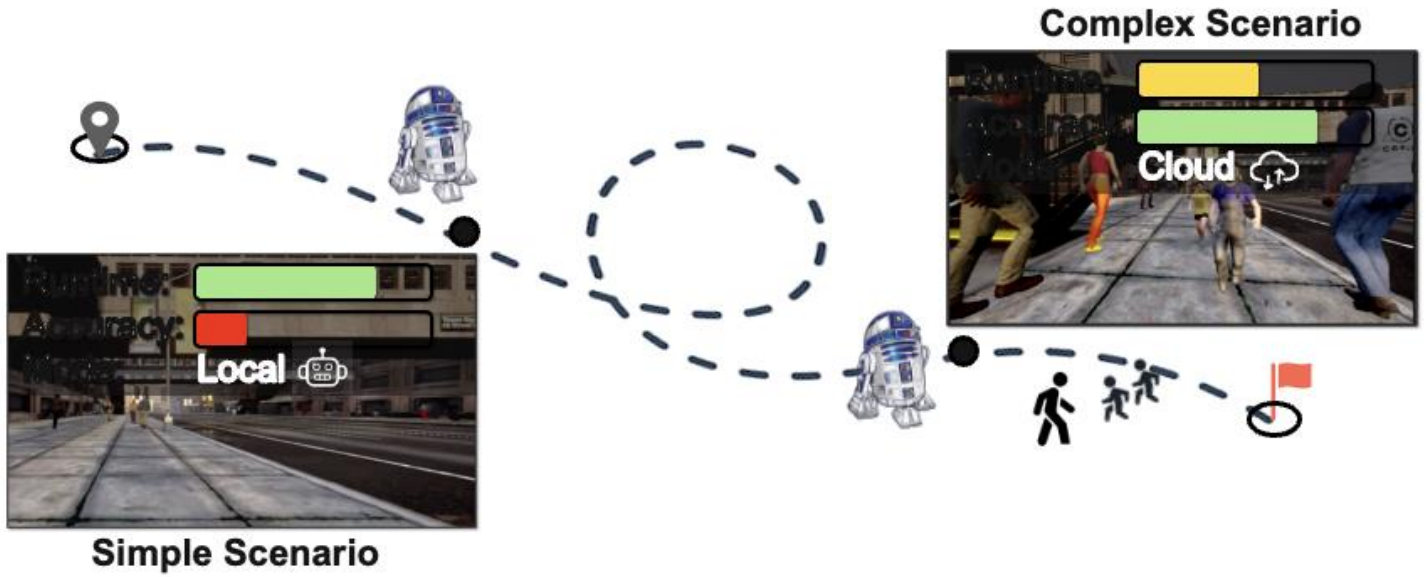
Staying On is better

- frequent switching is inefficient

If $T_{idle} > T_{BE}$

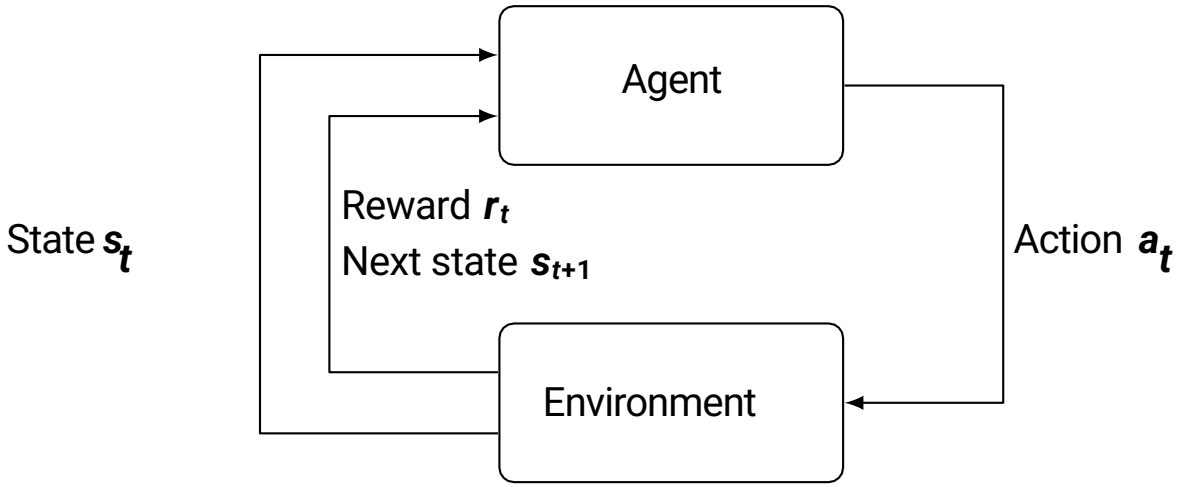
- system remains idle long enough to make up for the transition energy loss

Schedule Should Depend on the Situation!



We present UniLCD, a novel local-cloud hybrid framework that dynamically routes computation between low-power local models and powerful cloud resources via reinforcement learning based on scenario complexity.

Sequential Decision Process



Agent oberserves environment state s_t at time t

Agent sends action a_t at time t to the environment

Environment returns the reward r_t and its new state s_{t+1} to the agent

Decision Process

Components:

- State: $s \in \mathcal{S}$ — may be partially observed (e.g., game screen)
- Action: $a \in \mathcal{A}$ — may be discrete or continuous (e.g., turn angle, speed)
- Policy: $\pi_\theta : \mathcal{S} \to \mathcal{A}$ — we want to learn the policy parameters θ
- Optimal action: $a^* \in \mathcal{A}$ — provided by expert demonstrator
- Optimal policy: $\pi^* : \mathcal{S} \to \mathcal{A}$ — provided by expert demonstrator
- State dynamics: $P(s_{i+1}|s_i, a_i)$ — simulator, typically not known to policy
  Often deterministic: $s_{i+1} = T(s_i, a_i)$ — deterministic mapping
- Rollout: Given s_0, sequentially execute $a_i = \pi_\theta(s_i)$ & sample $s_{i+1} \sim P(s_{i+1}|s_i, a_i)$ yields trajectory $\tau = (s_0, a_0, s_1, a_1, \dots)$
- Loss function: $\mathcal{L}(a^*, a)$ — loss of action a given optimal action a^*

Decision Process

Decision Process (MDP) defined by tuple:

$$(S, A, R, P, \gamma)$$

- S : set of possible states
- A: set of possible actions
- R: distribution of reward given (state,action) pair
- P :distribution over next state given (state,action) pair
- γ: discount factor

Many reinforcement learning problems can be formalized as MDPs

Markov Decision Process

Markov property: Current state completely characterizes state of the world

- A state s_t is *Markov* if and only if

$$P(s_{t+1} \mid s_t) = P(s_{t+1} \mid s_1, \ldots, s_t)$$

- "The future is independent of the past given the present"
- The state captures all relevant information from the history
- Once the state is known, the history may be thrown away
- i.e. the state is a sufficient statistic of the future

Step 1

Collect demonstration data and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3.5 with supervised learning.

Step 2

Collect comparison data and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.

Step 3

Optimize a policy against the reward model using the PPO reinforcement learning algorithm.

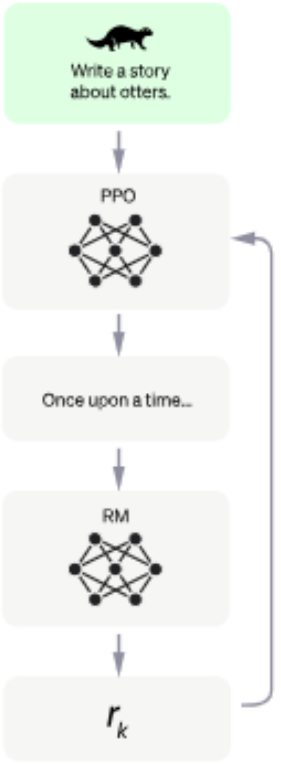
A new prompt is sampled from the dataset.

The PPO model is initialized from the supervised policy.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.



Step 1

Collect demonstration data and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3.5 with supervised learning.

Step 1

Collect demonstration data and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3.5 with supervised learning.

Step 2

Collect comparison data and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.

Step 1

Collect demonstration data and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3.5 with supervised learning.

Step 2

Collect comparison data and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.

Step 3

Optimize a policy against the reward model using the PPO reinforcement learning algorithm.

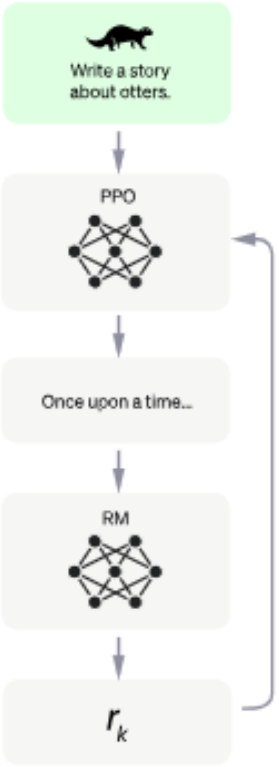
A new prompt is sampled from the dataset.

The PPO model is initialized from the supervised policy.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.



Step 1

Collect demonstration data and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3.5 with supervised learning.

Step 2

Collect comparison data and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.

Step 3

Optimize a policy against the reward model using the PPO reinforcement learning algorithm.

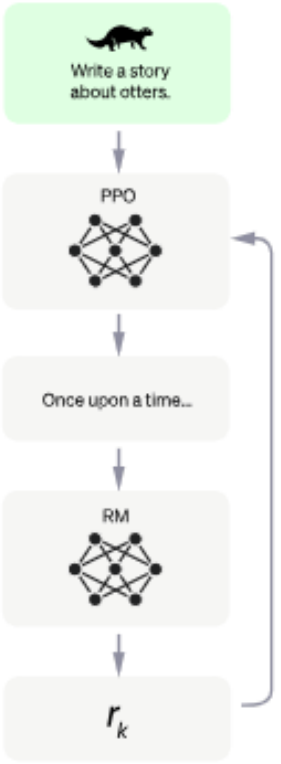
A new prompt is sampled from the dataset.

The PPO model is initialized from the supervised policy.

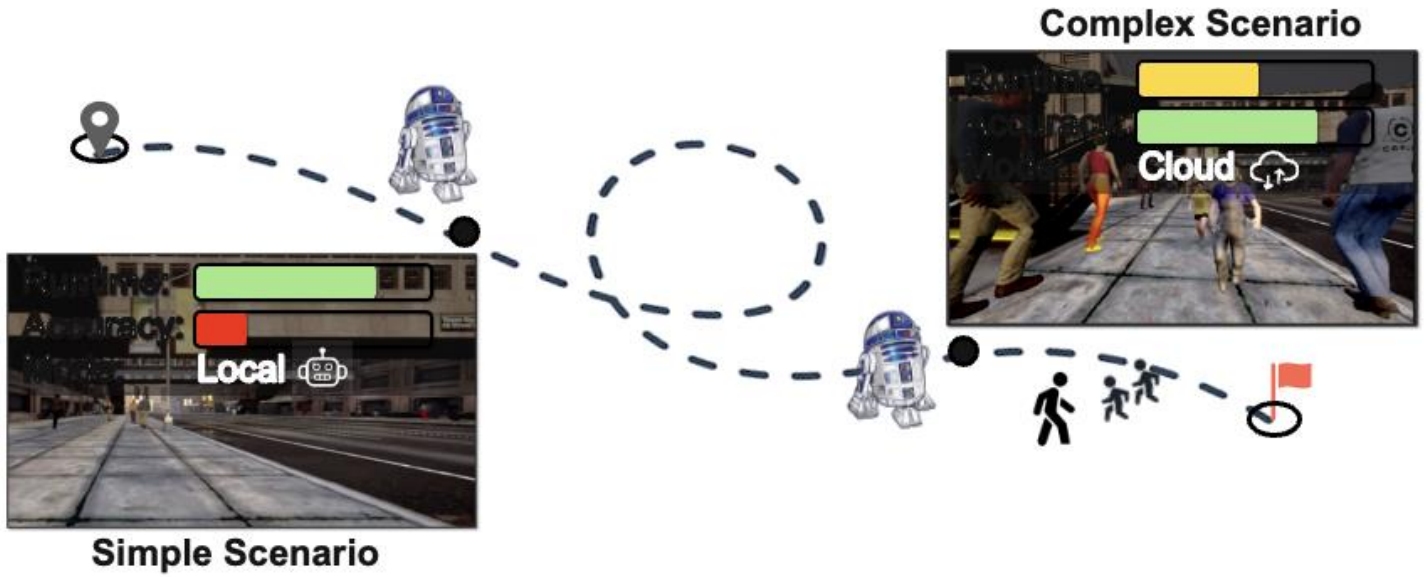
The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.



Schedule Should Depend on the Situation!



We present UniLCD, a novel local-cloud hybrid framework that dynamically routes computation between low-power local models and powerful cloud resources via reinforcement learning based on scenario complexity.

UniLCD framework

π_{φ}^{c} **Cloud Policy**

π_{θ}^{l} **Local Policy**

$\pi_{w}^{r}(\mathbf{o}|\ \pi_{\varphi}^{c}, \pi_{\theta}^{l})$ **Routing Policy**

$\mathbf{o} = \{\mathrm{I}, \mathrm{p}\}$ **Observations**

UniLCD framework

π_{φ}^{c} **Cloud Policy**

π_{θ}^{l} **Local Policy**

$\pi_{w}^{r}(\mathbf{o}|\ \pi_{\varphi}^{c}, \pi_{\theta}^{l})$ **Routing Policy**

$\mathbf{o} = \{\mathrm{I}, \mathrm{p}\}$ **Observations**

Goal $\mathbf{p}_t$

Image $\mathbf{I}_t$

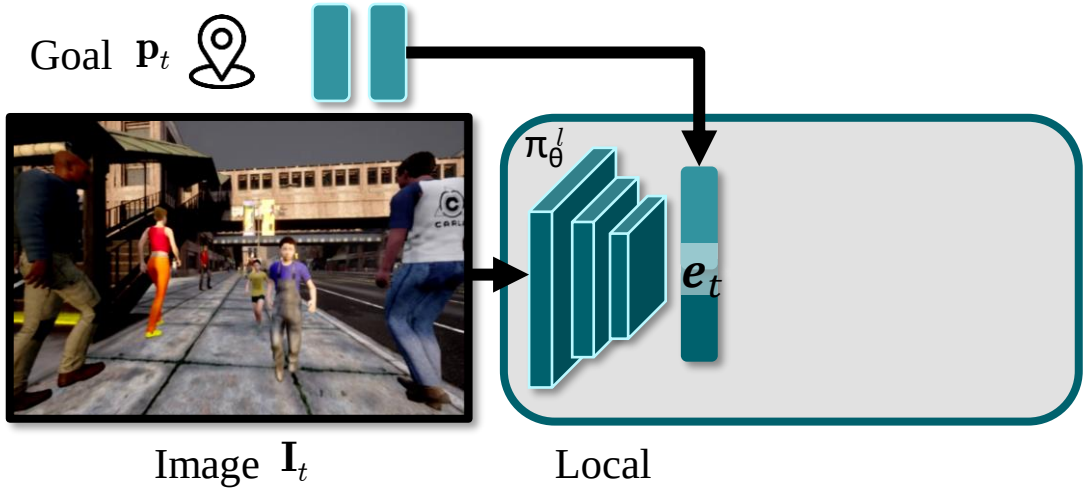
UniLCD framework

π_{φ}^{c} **Cloud Policy**

π_{θ}^{l} **Local Policy**

$\pi_{w}^{r}(\mathbf{o}|\ \pi_{\varphi}^{c}, \pi_{\theta}^{l})$ **Routing Policy**

$\mathbf{o} = \{\mathrm{I}, \mathrm{p}\}$ **Observations**



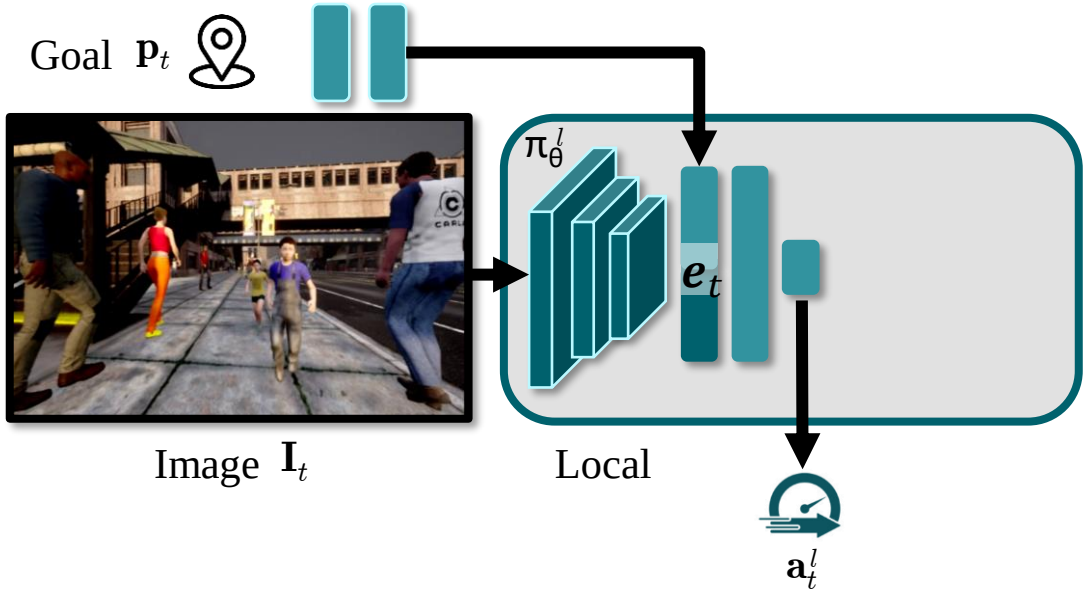
UniLCD framework

π_{φ}^{c} **Cloud Policy**

π_{θ}^{l} **Local Policy**

$\pi_{w}^{r}(\mathbf{o}|\ \pi_{\varphi}^{c}, \pi_{\theta}^{l})$ **Routing Policy**

$\mathbf{o} = \{\mathrm{I}, \mathrm{p}\}$ **Observations**



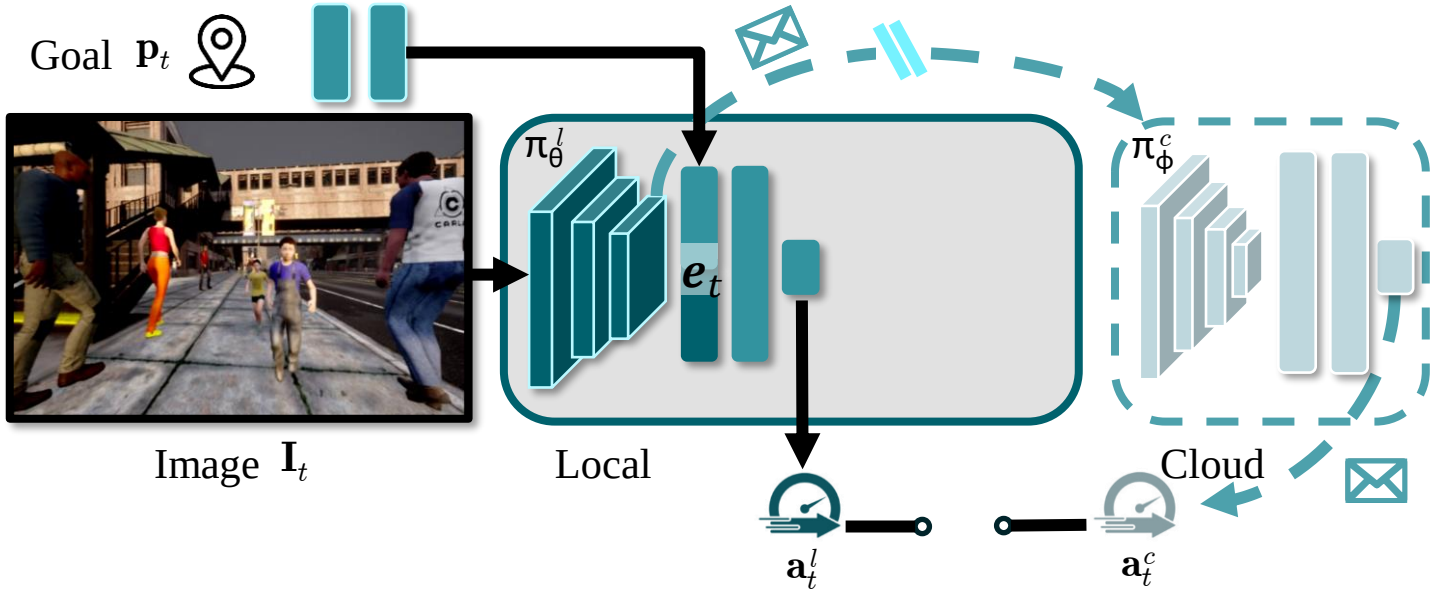
UniLCD framework

π_{φ}^{c} **Cloud Policy**

π_{θ}^{l} **Local Policy**

$\pi_{w}^{r}(\mathbf{o}|\ \pi_{\varphi}^{c}, \pi_{\theta}^{l})$ **Routing Policy**

$\mathbf{o} = \{\mathrm{I}, \mathrm{p}\}$ **Observations**



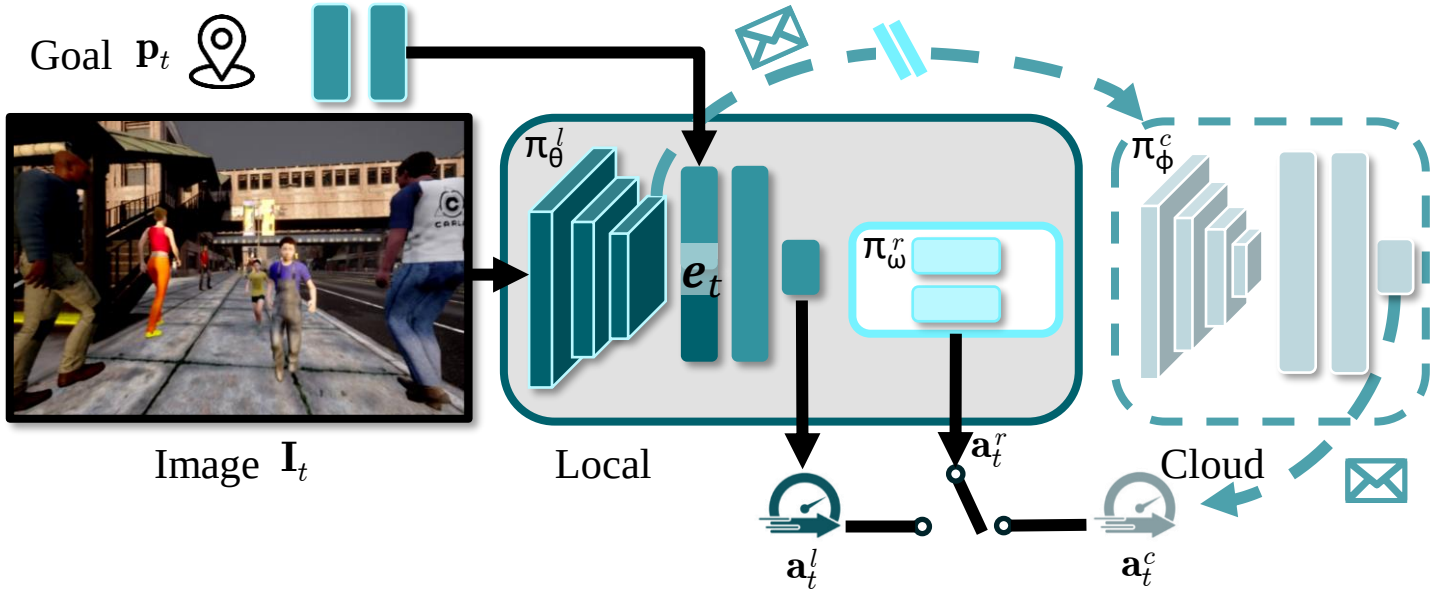
UniLCD framework

π_{φ}^{c} **Cloud Policy**

π_{θ}^{l} **Local Policy**

$\pi_{w}^{r}(\mathbf{o}|\,\pi_{\varphi}^{c}, \pi_{\theta}^{l})$ **Routing Policy**

$\mathbf{o} = \{\mathrm{I}, \mathrm{p}\}$ **Observations**



Multi-objective Reward Function

Task Reward: $r = (r_{geo}.r_{speed}.r_{energy}.r_{action})^{\alpha} - r_{collision}$

Geodesic Reward: $r_{geo} = (1 - \tanh(d_{geo}))$

Speed Reward: $r_{speed} = \frac{v}{m_v}$

Energy Disadvantage: $r_{energy} = 1 - \frac{e}{m_e}$

Extreme Action Clip: $r_{action} = \mathbb{I}(|r_{speed}| < \varepsilon).\mathbb{I}(\left|\frac{d}{d_m}\right| < \varepsilon)$

d_{geo} : Geodesic distance

m_v: **Maximum speed**

m_e: **Maximum energy**

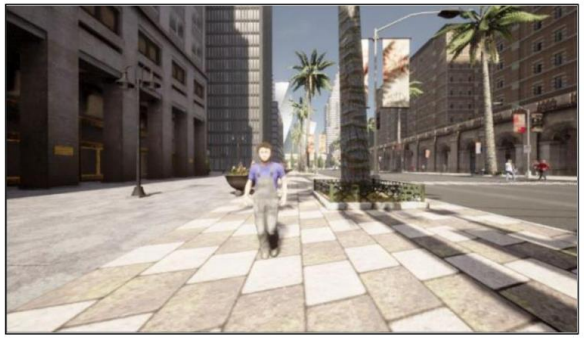
d_m : **Maximum possible rotation**

ε : **0.97 (Threshold for clipping action)**

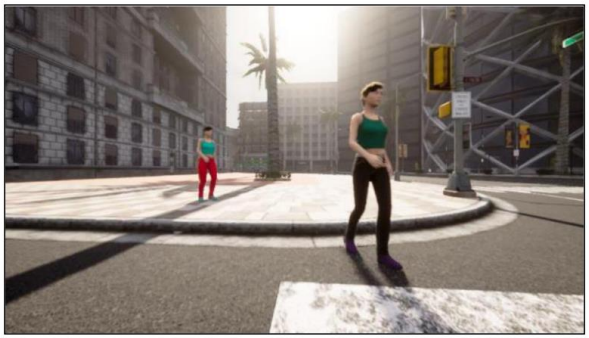
$r_{collision}$ **is a substantially high negative penalty to our robot every time it collides with a pedestrian.**

Simulation Environment

Low



Medium



High

Crowd

Comparing UniLCD with other baselines

Method	ENS↑	NS↑	SR↑	RC↑	Infract.↓	Energy↓	FPS↑
† Cloud-Only [84]	0.00	96.47	93.33	98.50	0.03	36.49	7.11
Local-Only [82]	63.43	67.33	0.00	75.23	0.16	4.33	65.40
Baseline Methods:							
Compressive Offloading [108]	13.98	80.16	0.00	80.16	0.00	90.66	1.82
† Selective Query [39]	24.14	61.28	0.00	82.68	0.11	45.35	18.14
† Adaptive Offloading [95]	37.42	40.37	70.00	94.05	1.22	4.80	30.14
Neurosurgeon [40]	39.85	63.10	0.00	80.54	0.03	28.31	12.53
SPINN [52]	36.31	72.75	60.00	92.73	0.35	18.94	20.37
Deep Sequential RL [97]	58.84	61.83	0.00	79.36	0.36	3.77	77.94
UniLCD Module Ablations:							
† Standard Reward	48.35	54.99	0.00	75.23	0.13	3.57	50.20
† Standard Reward w/ History	50.04	57.21	10.00	77.71	0.12	8.38	49.07
† Our Reward (Eq. (2))	48.30	79.90	56.66	91.15	0.19	21.72	16.05
† Our Reward w/ History	71.70	87.71	83.33	94.66	0.11	7.83	33.98
Our Reward (Eq. (2))	57.20	87.39	60.00	91.10	0.06	6.60	**12.49**
Our Reward w/ History	**85.97**	**94.58**	**93.33**	**95.90**	**0.02**	**2.90**	26.49

Qualitative Results

Exercise -- March 19, 2025
Dataflow design in SW

Recall synchronous dataflow (SDF) models from lectures. In this exercise, your task is to design a dataflow system that prints the multiples of 24 (24, 24*2, 24*3, 24*4, ...) using the following actors:

- The *fork* actor is a single-rate actor with one input and two outputs. Each input token is read, duplicated and copied to both outputs.
- The *increment* actor is a single-rate actor with one input and one output. An input token is read, incremented and copied to the output.
- The *multiply* actor is a single-rate actor with two inputs and one output. One token is read from each input, their token values are multiplied, and the result is copied to the output.
- The *print* actor is a single-rate actor with one input. It prints the value of the input token.

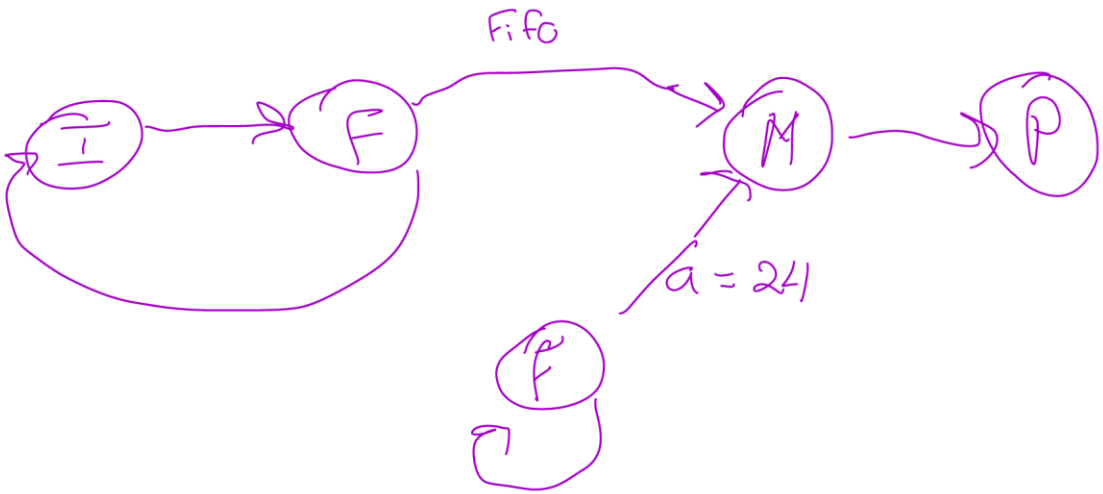
You will be developing the C code for each actor. A FIFO library to help you implement dataflow queues is provided for you. As an example, we will first implement the code for the multiply actor. Let's first consider fifo.h, the interface for the dataflow queue.

Exercise -- March 19, 2025
Dataflow design in SW

Recall synchronous dataflow (SDF) models from lectures. In this exercise, your task is to design a dataflow system that prints the multiples of 24 (24, 24*2, 24*3, 24*4, ...) using the following actors:

- The *fork* actor is a single-rate actor with one input and two outputs. Each input token is read, duplicated and copied to both outputs.
- The *increment* actor is a single-rate actor with one input and one output. An input token is read, incremented and copied to the output.
- The *multiply* actor is a single-rate actor with two inputs and one output. One token is read from each input, their token values are multiplied, and the result is copied to the output.
- The *print* actor is a single-rate actor with one input. It prints the value of the input token.

You will be developing the C code for each actor. A FIFO library to help you implement dataflow queues is provided for you. As an example, we will first implement the code for the multiply actor. Let's first consider fifo.h, the interface for the dataflow queue.



```
#ifndef ACT_PRINT_H
#define ACT_PRINT_H
#include "fifo.h“
#include <stdio.h>
#include <assert.h>

void actor_mul(fifo_t *F1, fifo_t *F2, fifo_t *q);
void actor_print(fifo_t *F);
void actor_fork(fifo_t *F1, fifo_t *q1, fifo_t *q2);

#endif
```

```
#include "actors.h"

void actor_mul(fifo_t *F1, fifo_t *F2, fifo_t *q)
{
        assert(F1 != 0);
        assert(F2 != 0);
        assert(q != 0);
        // firing rule: fire if there's at least one token on each of the inputs, F1 and F2
        if ((fifo_size(F1) > 0) && (fifo_size(F2) > 0))
                put_fifo(q, get_fifo(F1) * get_fifo(F2));
}

void actor_print(fifo_t *F)
{
        assert(F!=0);
        //firing rule: fire if there's at least one token in F
        if (fifo_size(F) > 0)
                printf("%d\n", get_fifo(F));
}

void actor_increment(fifo_t *F, fifo_t *F2){
        assert(F!=0);
        if (fifo_size(F) > 0){
                put_fifo(F2, get_fifo(F)+1);
        }
}

void actor_fork(fifo_t *F1, fifo_t *q1, fifo_t *q2)
{
   assert(F1 != 0);
   assert(q1 != 0);
   assert(q2 != 0);
  int temp = get_fifo(F1);
  //Beware of pointers!
  if ((fifo_size(F1) > 0))
  {
    put_fifo(q1, temp);
    put_fifo(q2, temp);
  }
}
```

```
#ifndef FIFO_H
#define FIFO_H

#define MAXFIFO 1024

typedef struct fifo {
  int data[MAXFIFO];
  unsigned wptr;
  unsigned rptr;
} fifo_t;

void     init_fifo(fifo_t *F);
void     put_fifo (fifo_t *F, int d);
int      get_fifo (fifo_t *F);
unsigned fifo_size(fifo_t *F);

void actor_increment(fifo_t *F, fifo_t *F2);

#endif
```

```
#include "fifo.h"
#include <assert.h>

void init_fifo(fifo_t *F) {
  F->wptr = F->rptr = 0;
}

void put_fifo(fifo_t *F, int d) {
  if (((F->wptr + 1) % MAXFIFO) != F->rptr) {
    F->data[F->wptr] = d;
    F->wptr = (F->wptr + 1) % MAXFIFO;
    assert(fifo_size(F) <= 10);
  }
}

int get_fifo(fifo_t *F) {
  int r;
  if (F->rptr != F->wptr) {
    r = F->data[F->rptr];
    F->rptr = (F->rptr + 1) % MAXFIFO;
    return r;
  }
  return -1;
}

unsigned fifo_size(fifo_t *F) {
  if (F->wptr >= F->rptr)
    return F->wptr - F->rptr;
  else
    return MAXFIFO - (F->rptr - F->wptr) + 1;
}
```