Context-Aware Task Handling in Resource-Constrained Robots with Virtualization

IEEE Edge'23
July 2023

Ramyad Hadidi* *Rain AI*

Nima Shoghi *Georgia Tech*

Bahar Asgari* *University of Maryland*

Hyesoon Kim *Georgia Tech*

* This work was done when the authors were affiliated with Georgia Tech.

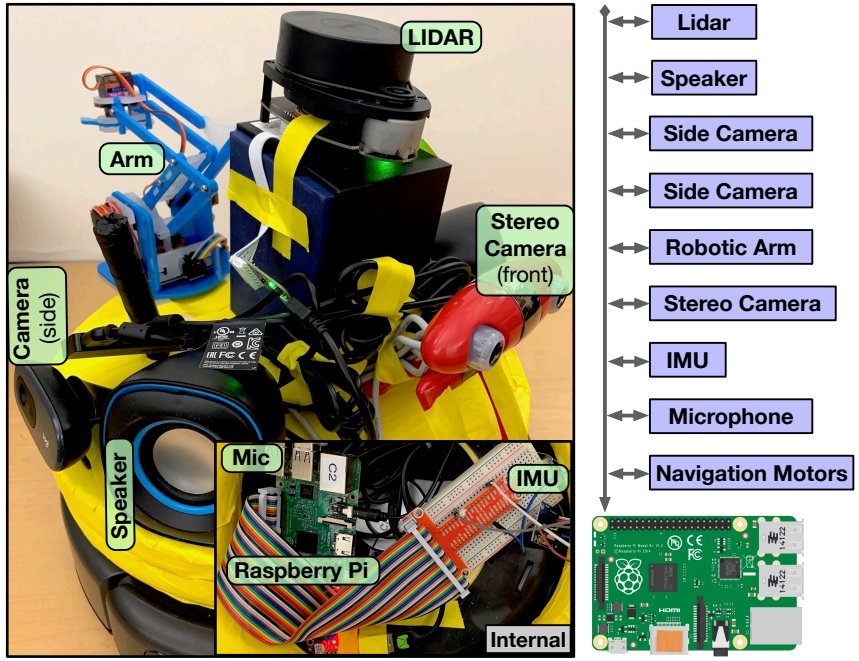
Georgia Tech

Mobile Robots

Mobile robots are used in various scenarios:

- Interacting with a complex and non-deterministic world
- Executing numerous tasks like

  Sensing,
  planning,
  manipulating,
  and reasoning

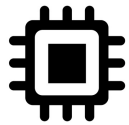


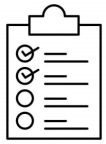
Our Raspberry-Pi-Based Robot

The Challenge of Mobile Robots

In a complex world, all intelligent mobile robots are in a **never-ending conflict** among:

- Limited computational resources
- Energy storage, and
- Pending tasks requiring resources



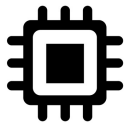


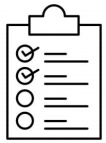
Resource-constrained mobile robots have even more challenges to meet all the real-time deadlines

The Challenge of Mobile Robots

In a complex world, all intelligent mobile robots are in a **never-ending conflict** among:

- Limited computational resources
- Energy storage, and
- Pending tasks requiring resources





Resource-constrained mobile robots have even more challenges to meet all the real-time deadlines

Need for solutions that ensure **real-time** performance with **concurrent task handling**

Current Solutions and Their Limitations

First group involves adding extra hardware or utilizing cloud/fog computation

- Adding extra hardware is infeasible or uneconomical
  - The most common approach:

    Dedicate one processor per task! Not scalable
  - Adding extra hardware is often infeasible
    For instance, in a lightweight drone
- Cloud and fog computations are not always available and privacy concerns limit their use

Current Solutions and Their Limitations

Second group involves Real-Time OS schedulers or Robotic OS (ROS)

- Real-time OS schedulers are designed to minimize latency with preemptive schedulers
  - They cannot guarantee hard deadlines
  - They are not dynamic because fixed schedulers
- ROS does not offer real-time operations
- ROS2 does not support dynamically changing priorities in runtime and requires additional Linux kernel support

Solution – Example

Detecting contexts to improve real timeliness of critical tasks by dynamically reducing number of tasks in runtime

Solution – Example

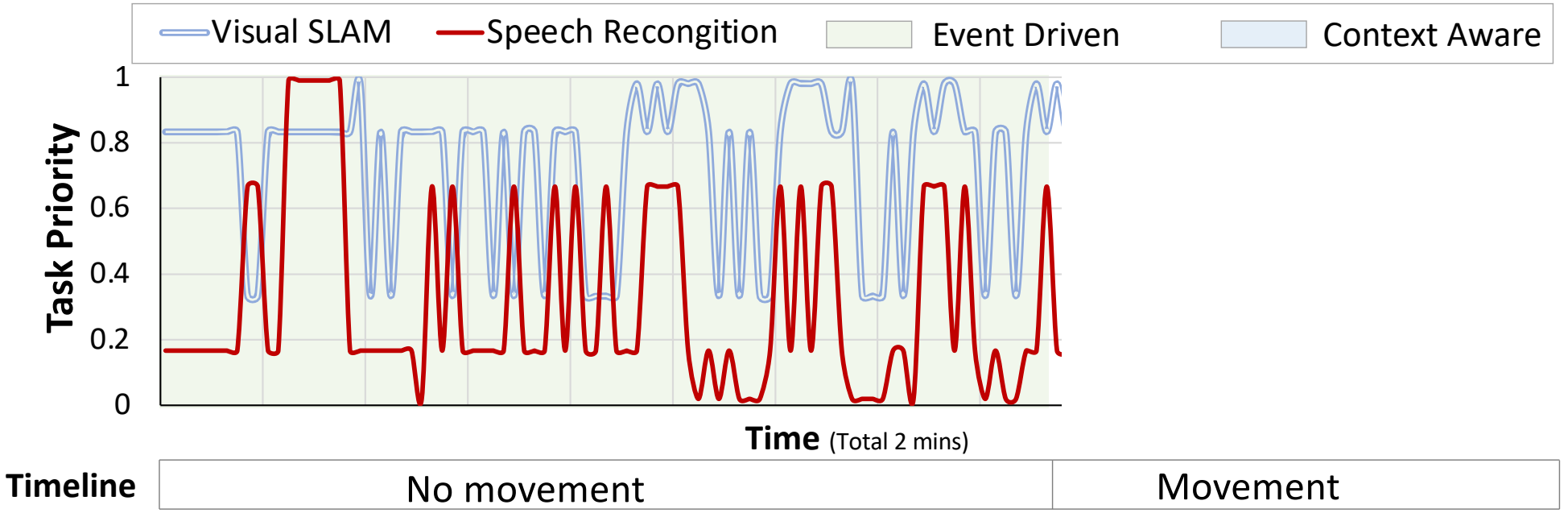
Detecting contexts to improve real timeliness of critical tasks by dynamically reducing number of tasks in runtime

Simple Example of a speech enabled vacuum robot

Solution – Example

Detecting contexts to improve real timeliness of critical tasks by dynamically reducing number of tasks in runtime

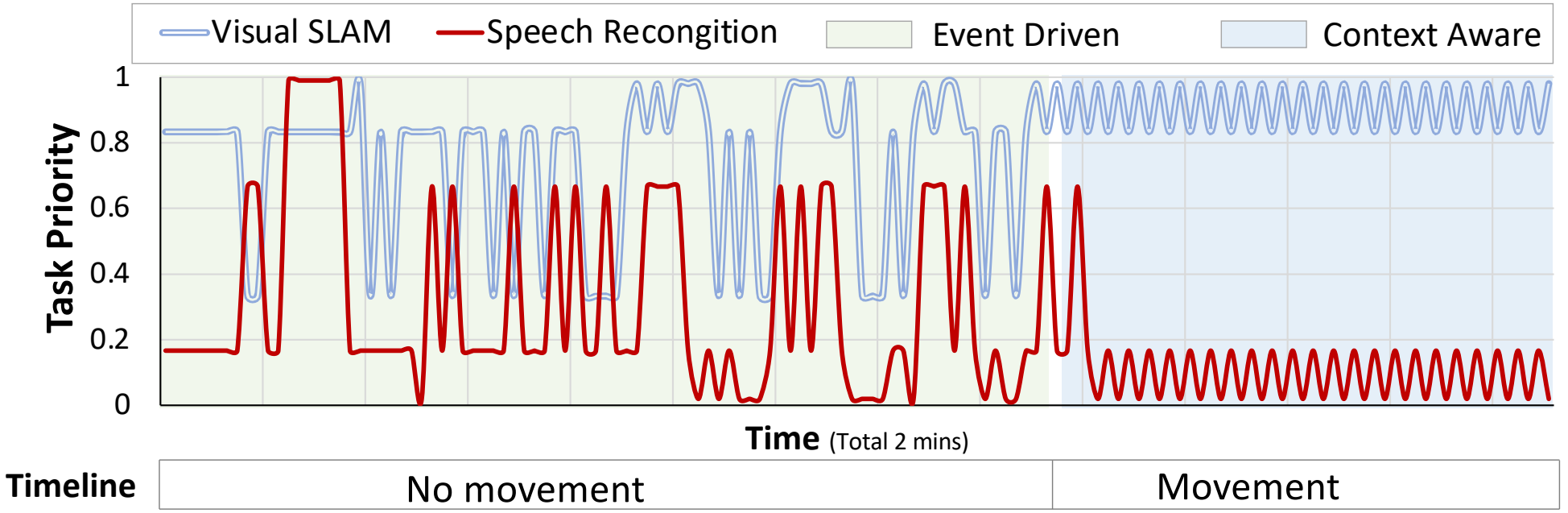
Simple Example of a speech enabled vacuum robot



Solution – Example

Detecting contexts to improve real timeliness of critical tasks by dynamically reducing number of tasks in runtime

Simple Example of a speech enabled vacuum robot



Solution

Context-Aware Task Handling — to dynamically simplify world for easier planning

OS-Level Dynamic Time Sharing — to meet hard real-time deadlines

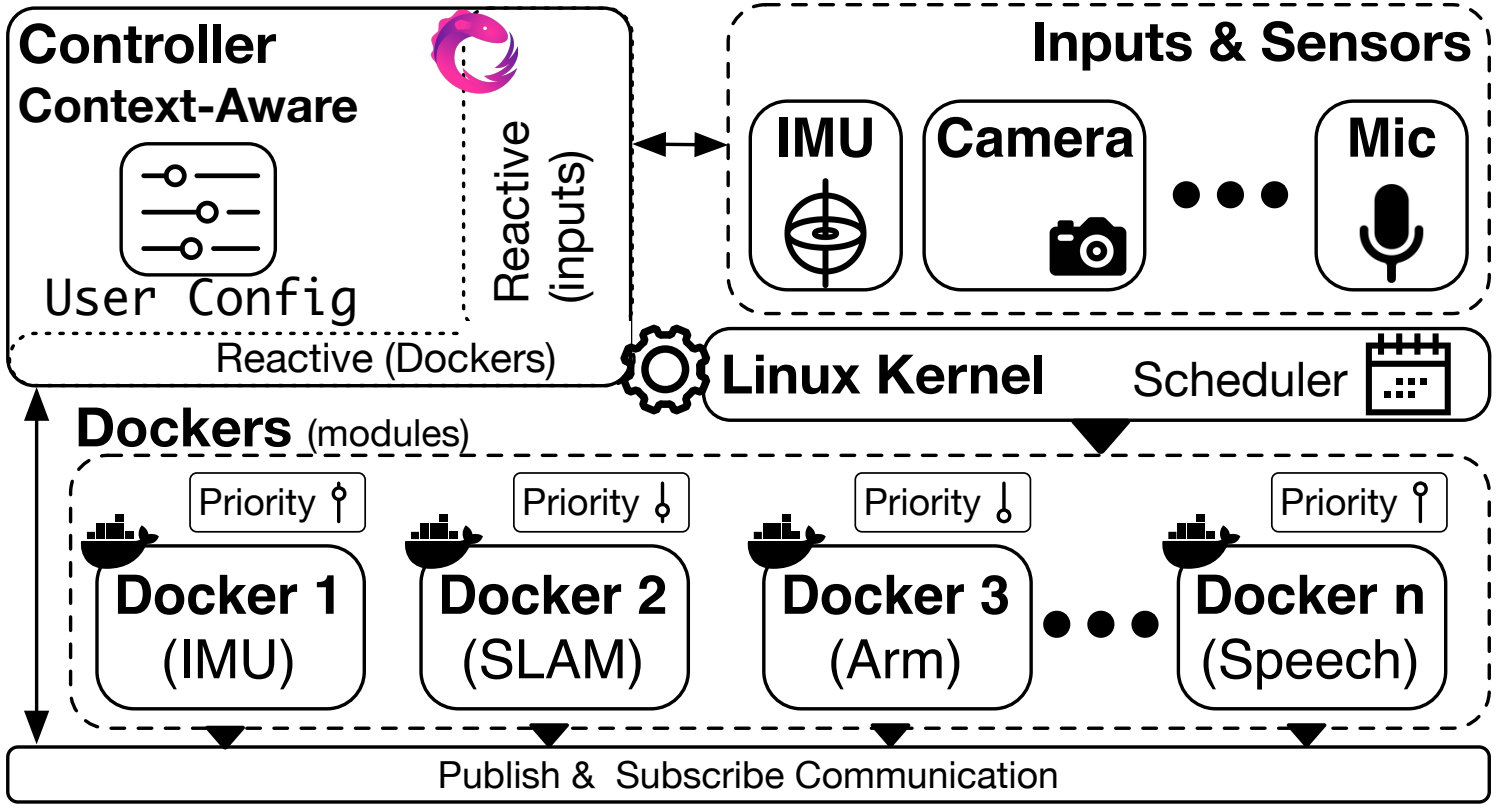
Event-Driven Scheduling — to mindfully use limited resources

Virtualized Execution — to implement a lightweight and programmable reactive paradigm + no Linux kernel modifications

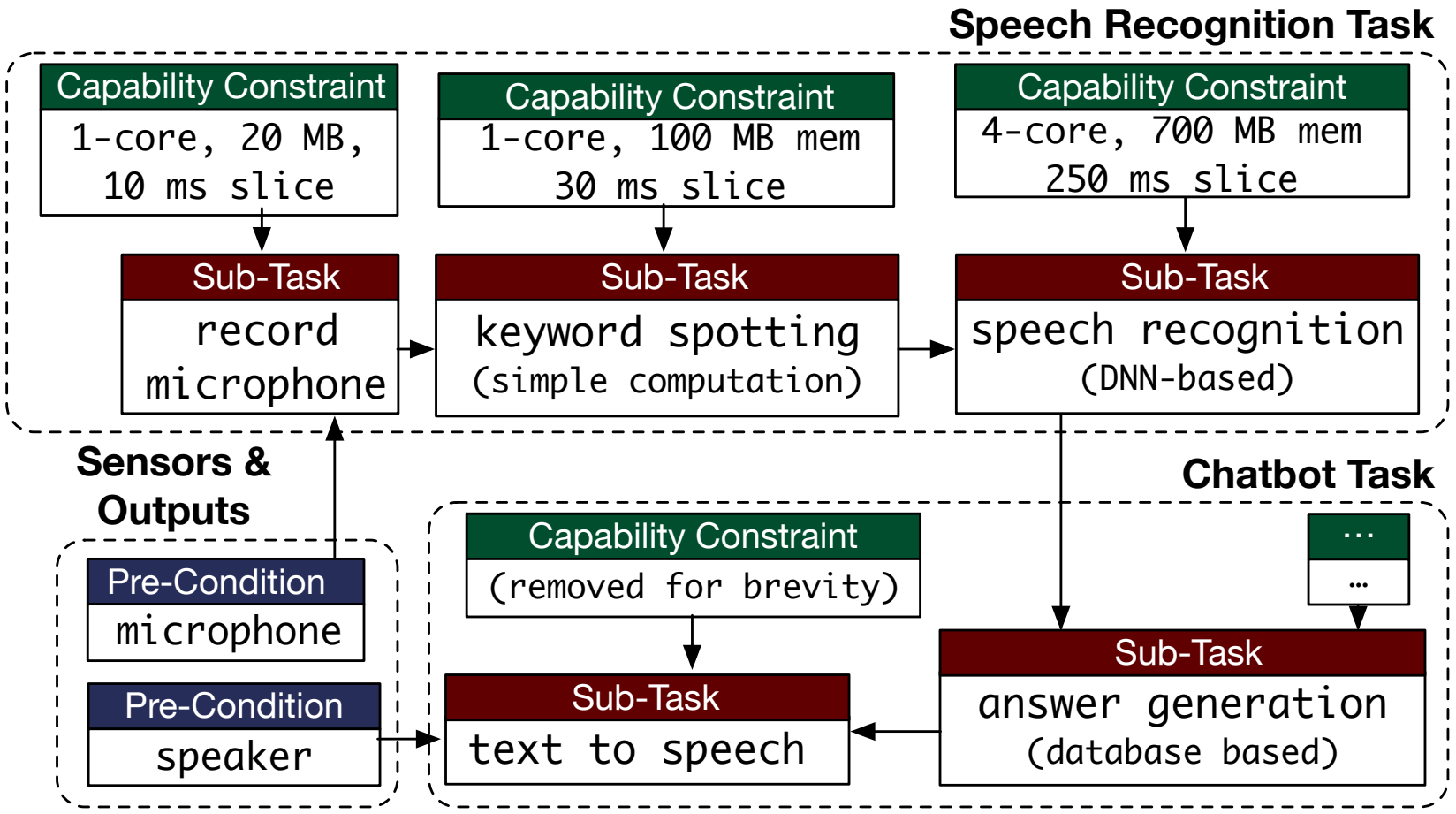
System Overview

Multiple components at play:

Task graphs | Docker/task | Context-Aware controller



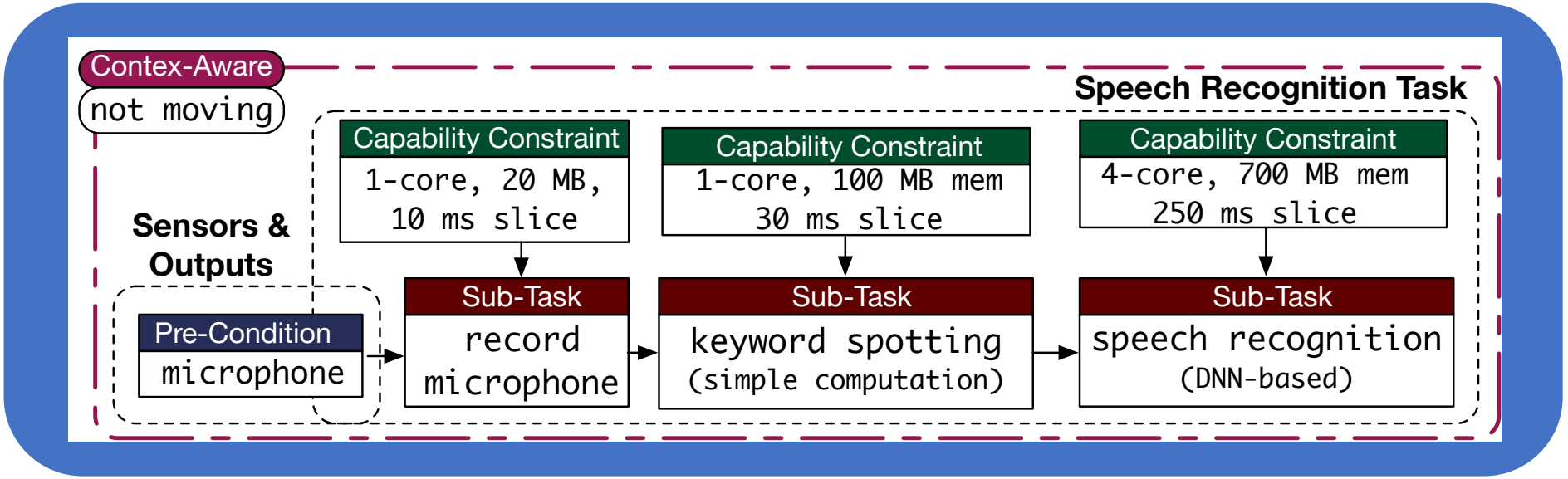
Task Graphs


Speech Recognition Task
Capability Constraint
1-core, 20 MB,
10 ms slice
Capability Constraint
1-core, 100 MB mem
30 ms slice
Capability Constraint
4-core, 700 MB mem
250 ms slice
Sub-Task
record
microphone
Sub-Task
keyword spotting
(simple computation)
Sub-Task
speech recognition
(DNN-based)
Sensors &
Outputs
Chatbot Task
Capability Constraint
(removed for brevity)
…
…
Pre-Condition
microphone
Pre-Condition
speaker
Sub-Task
text to speech
Sub-Task
answer generation
(database based)

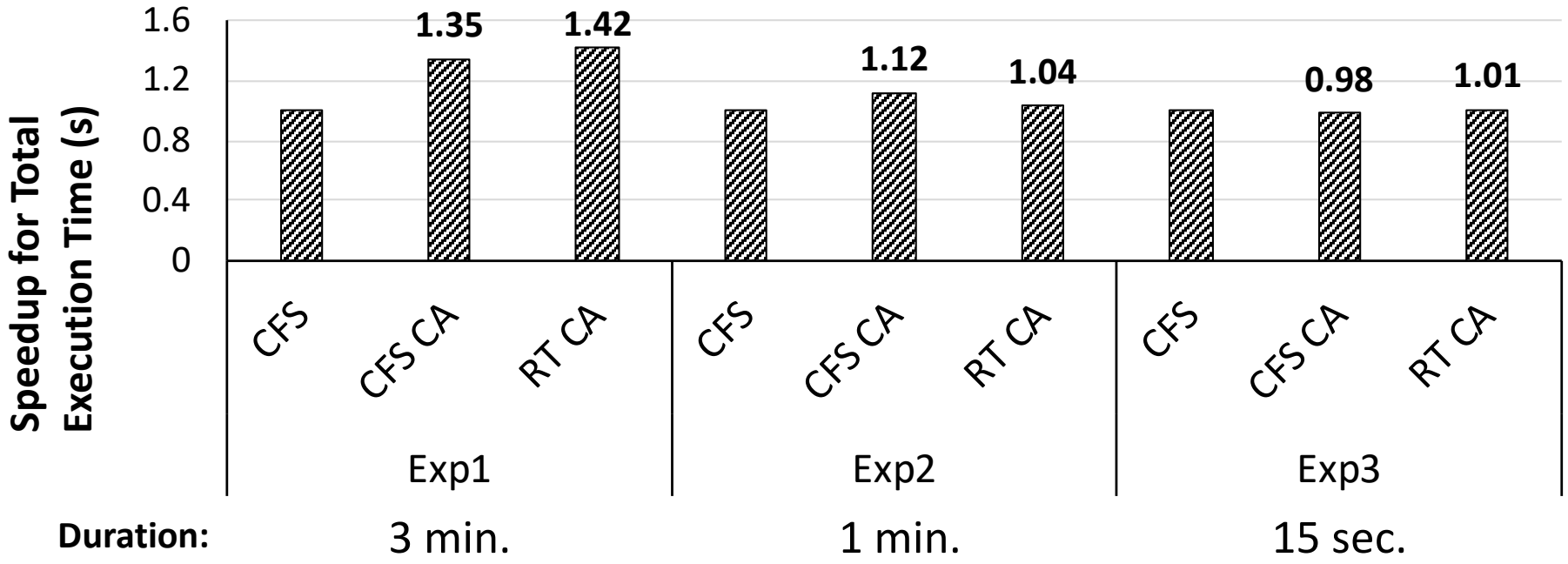
Task Graphs with Reactive Paradigm

An example with IMU sensor:

```
imu.pipe(filter(lambda value:                                    1
    value["accelerometer"]["x"] != 0 and                         2
    value["accelerometer"]["y"] != 0 and                         3
    value["accelerometer"]["z"] != 0))                           4
```



Results



Legend -- CFS: Linux Scheduler

-- CFS CA: Linux Scheduler + Context Aware

-- RT CA: Real-Time Scheduler + Context Aware

Note -- Non-Context-Aware settings miss deadlines!


Context-Aware Task Handling in
Resource-Constrained Robots with
Virtualized Execution

Please check the paper for more details on

- Scheduling policies,
- Integration with Linux,
- Docker implementation,
- Experiments details, and
- Planning algorithm

Context-Aware Task Handling in Resource-Constrained Robots with Virtualization

Ramyad Hadidi[§] — *Rain AI* — ramyad@rain.ai

Nima Shoghi Ghaleshahi — *Georgia Tech* — nimash@gatech.edu

Bahar Asgari[§] — *University of Maryland* — bahar@umd.edu

Hyesoon Kim — *Georgia Tech* — hyesoon.kim@gatech.edu

***Abstract*—Intelligent mobile robots are critical in several scenarios. However, as their computational resources are limited, mobile robots struggle to handle several tasks concurrently while guaranteeing real timeliness. To address this challenge and improve the real-timeliness of critical tasks under resource constraints, we propose a *fast context-aware* task handling technique. To effectively handle tasks in *real-time*, our proposed context-aware technique comprises three main ingredients: (i) a dynamic time-sharing mechanism, coupled with (ii) an event-driven task scheduling using reactive programming paradigm to mindfully use the limited resources; and, (iii) a lightweight virtualized execution to easily integrate functionalities and their dependencies. We showcase our technique on a Raspberry-Pi-based robot with a variety of tasks such as Simultaneous localization and mapping (SLAM), sign detection, and speech recognition with a 42% speedup in total execution time compared to the common Linux scheduler.**

***Index Terms*—Edge AI, Software, Mobile Robots, Middleware and Programming Environments, Reactive and Sensor-Based Planning,**

I. Introduction & Motivation

Unlike conventional industrial or commercialized robots that perform a set of pre-programmed and routine tasks, intelligent mobile robots manipulate their environment using their perception and physical resources to achieve a myriad of goals. Such robots must be capable of dynamically switching between navigation, planning, reasoning, recognition, and sensing their environment. Intelligent robots need to interact with a dynamic, complex, and non-deterministic world. These robots must execute numerous tasks such as controlling their physical resources (*e.g.*, arms), understanding data derived from sensors, or executing perception and planning.

Intelligent robots are always in a never-ending conflict between available computation resources, their energy storage, and the tasks at hand. This conflict is particularly emphasized in resource-constrained robots because even the concurrent execution of a few rudimentary tasks is extremely demanding with only a few processing cores. For example, a Raspberry Pi with only four cores could be fully utilized by the operation system (OS), processing the data from a single sensor, and simple navigation and control algorithms. Adding more sensors and tasks only causes the robot to miss real-time deadlines. Thus, ensuring efficient handling of critical tasks and meeting critical deadlines is the key challenge for resource-constrained robots.

To extend the capabilities of resource-constrained robots and meet real-time demands, the common practices are adding extra hardware or utilizing cloud/fog computation [1]–[7]. However, in several scenarios, adding new hardware is either infeasible or uneconomical. For example, adding extra processing units to a lightweight drone requires heavier batteries, which in turn demands stronger motors. Further, cloud and fog are not always available. Additionally, privacy concerns limit the suitability of cloud-based computation.

To enable intelligent mobile robots to efficiently utilize limited resources, we propose a *context-aware* task handling technique that simplifies the world and planning tasks by dynamically reducing the number of tasks in a certain context to only the critical ones. For example, limited human-robot interaction is expected while the robot is performing an already assigned task. This technique enables resource-constrained robots to efficiently perform manifold functionalities while meeting their real-time constraints.

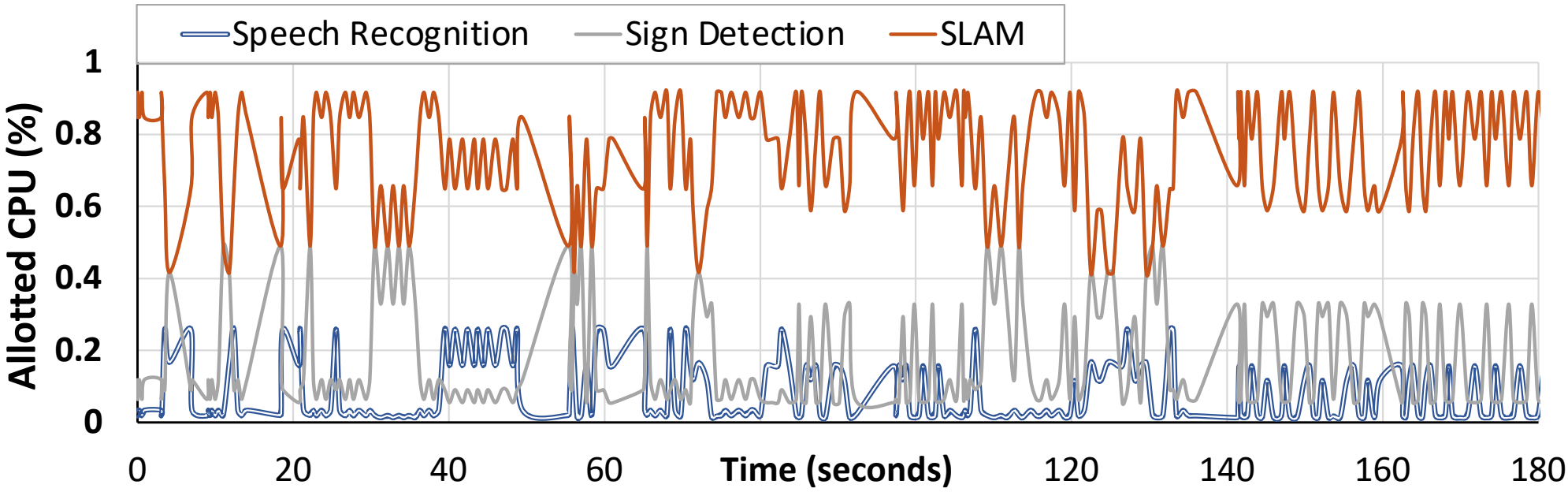
To be effective in handling tasks using our context-aware technique, we propose using a *virtualized execution* that (i) integrates several tasks while providing dynamic, low-cost, and kernel-level control over the scheduling policy; (ii) enables easier context-aware implementation by providing manageable control over tasks; and (iii) provides a uniform and practical environment for building new robots in the community.

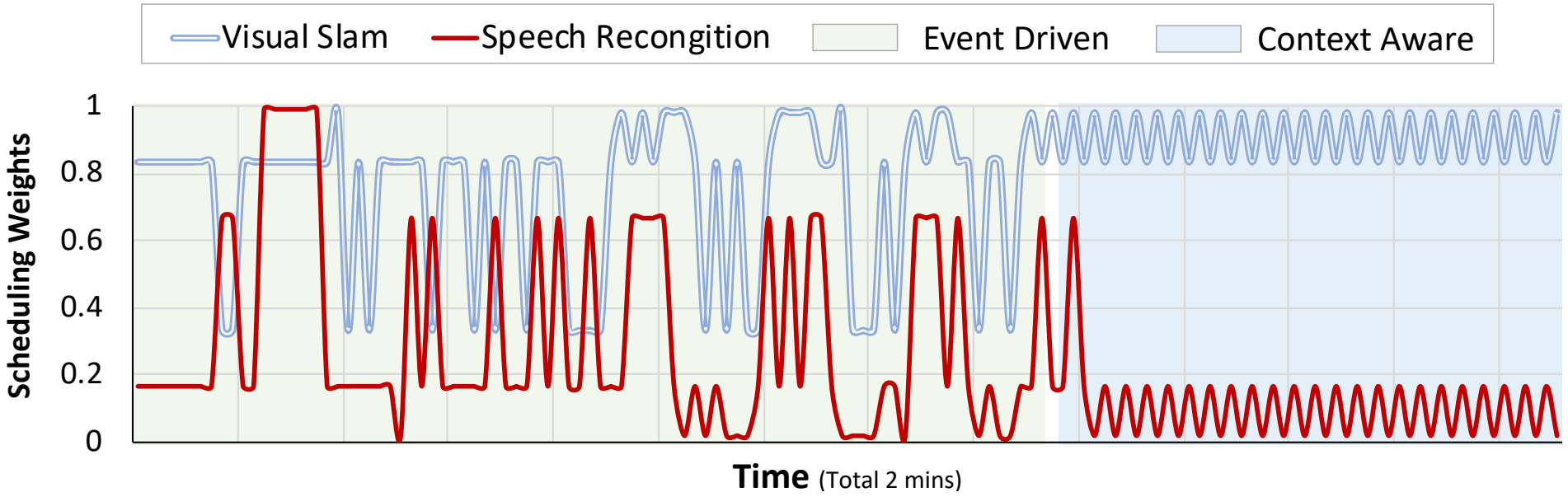
For experiments, we use a custom-built Raspberry-Pi-based robot using an iRobot Roomba [8] equipped with one Raspberry Pi 4 (RPi4) [9] as the only processing unit. Our iRobot, shown in Figure 1, has several sensors (*i.e.*, LIDAR, inertial measurement unit (IMU), cameras, and microphone), and control devices (*i.e.*, motors for navigation, robotic arm, and speakers). For software, we use Docker [10], a popular virtualization tool, and implement our context-aware technique to collect and process sensor data, simultaneous localization and mapping (SLAM), voice recognition, and sign recognition. Our contributions are as follows:

- *Context-aware* task planning to effectively use the limited resources and hence extend the number of tasks that a robot can handle.
- OS-level *dynamic* time-sharing to implement the context-aware scheduling in real-time.

This work was supported in part by the NSF grant number 2103951.

[§]This work was done when the authors were affiliated with Georgia Tech.


Speech Recognition
Sign Detection
SLAM
Allotted CPU (%)
1
0.8
0.6
0.4
0.2
0
0
20
40
60
Time (seconds)
120
140
160
180


Visual Slam
Speech Recongition
Event Driven
Context Aware
Scheduling Weights
1
0.8
0.6
0.4
0.2
0
Time (Total 2 mins)

Procedure 1: Context-Aware Task Planning.

```
   Input  : ConfigFile: Configuration File with Context-Aware Setting
            and Relative Priorities of Tasks.
   Input  : InputList: Input & Sensor List
1  Initial
2      for input ∈ InputList do
           // Create observable stream.
3          stream ← CreateStream(input);
           // Instantiate scheduling score function.
4          InstantiateRX(stream);
5          AddToList(stream, StreamList);
6      for context ∈ ConfigFile do
           // Create context task graph.
7          graph ← CreateTaskGraph(context);
           // Create combinator observable stream
8          contextStream ← CreateStream(graph);
9          AddToList(contextStream, TaskGraphList) ;
       // Initialize an initial scheduling policy.
10     InitializeScheduling ;
11     return StreamList TaskGraphList
   // On receiving an event on any stream after its bound
      reactive function.
12 Event-Based Procedure
13     OnObservableStreamEvent stream
           // Calculate scheduling score for the stream.
14         CalculateScore(stream) ;
           // Calculate new real-time time-slices values.
15         DictTimeSlices ← CalculateTimeSlices() ;
           // Update the scheduler.
16         UpdateScheduler(DictTimeSlices);
```