E2M: An Energy-Efficient Middleware for Computer Vision Applications on Autonomous Mobile Robots

Liangkai Liu
Wayne State University

Jiamin Chen
Wayne State University

Marco Brocanelli
Wayne State University

Weisong Shi
Wayne State University

ABSTRACT

Autonomous mobile robots (AMRs) have been widely utilized in industry to execute various on-board computer-vision applications including autonomous guidance, security patrol, object detection, and face recognition. Most of the applications executed by an AMR involve the analysis of camera images through trained machine learning models. Many research studies on machine learning focus either on performance without considering energy efficiency or on techniques such as pruning and compression to make the model more energy-efficient. However, most previous work do not study the root causes of energy inefficiency for the execution of those applications on AMRs. The computing stack on an AMR accounts for 33% of the total energy consumption and can thus highly impact the battery life of the robot. Because recharging an AMR may disrupt the application execution, it is important to efficiently utilize the available energy for maximized battery life.

In this paper, we first analyze the breakdown of power dissipation for the execution of computer-vision applications on AMRs and discover three main root causes of energy inefficiency: uncoordinated access to sensor data, performance-oriented model inference execution, and uncoordinated execution of concurrent jobs. In order to fix these three inefficiencies, we propose E2M, an energy-efficient middleware software stack for autonomous mobile robots. First, E2M regulates the access of different processes to sensor data, e.g., camera frames, so that the amount of data actually captured by concurrently executing jobs can be minimized. Second, based on a predefined per-process performance metric (e.g., safety, accuracy) and desired target, E2M manipulates the process execution period to find the best energy-performance trade off. Third, E2M coordinates the execution of the concurrent processes to maximize the total contiguous sleep time of the computing hardware for maximized energy savings. We have implemented a prototype of E2M on a real-world AMR. Our experimental results show that, compared to several baselines, E2M leads to 24% energy savings for the computing platform, which translates into an extra 11.5% of battery time and 14 extra minutes of robot runtime, with a performance degradation lower than 7.9% for safety and 1.84% for accuracy.

SEC 2019, November 7–9, 2019, Arlington, VA, USA

ACM ISBN 978-1-4503-6733-2/19/11…$15.00
https://doi.org/10.1145/3318216.3363302

1 INTRODUCTION

An Autonomous Mobile Robot (AMR) is mainly composed of mechanical parts (e.g., wheels, engine, etc.), sensors (e.g., camera), and computing hardware (e.g., CPU, GPU) that allow the robot to autonomously drive and perform a pre-defined set of jobs. Due to their relatively low cost, high flexibility, and high reliability, various AMRs have been designed and employed in several industry applications [4, 10, 25]. For example, an AMR can be programmed to patrol the fences of a private area so that the security team can be timely notified of intrusions. Typical jobs for AMRs include security patrol, object detection, and face recognition [40]. However, the correct operation of an AMR is strictly dependent on its limited battery life. Therefore, it is important to ensure high energy efficiency to maximize the battery life of AMRs.

Some previous studies have focused on reducing the energy consumption due to the mechanical parts of the AMR. For example, they propose solutions to dynamically find the most energy-efficient path to reach a certain location [3, 5, 8, 15]. However, these solutions do not take into consideration the computational portion of the AMR's energy consumption. In fact, as we show in Section 3, the computing resources of a typical AMR can account for the 33% of the total energy consumption. The most common applications of AMRs use machine learning models in computer-vision applications. Thus, in order to improve the energy efficiency of the computing resources of an AMR, it is important to focus on the execution of computer-vision based applications. Unfortunately, most previous research studies on such applications either focus mainly on performance [28, 29, 42] or employ pruning and compression techniques to make the trained model more energy efficient [46]. To the best of our knowledge, no previous work has yet studied the energy efficiency of AMRs during the execution of computer-vision based applications.

Due to the above-described shortcomings of existing literature on AMRs, in this paper we first conduct an in-depth study of the computer-vision application execution on AMRs to profile its energy consumption and to discover the main sources of inefficiency. We find two main sources of high energy consumption across various applications: access to sensor data and model inference. Accordingly, we find **three main inefficiencies** for these two energy sources. First, **uncoordinated access to sensor data.** Each computer-vision process directly interacts with the sensors, e.g., camera, of the AMR to acquire data, e.g., camera frames. As a result, N concurrently executing processes may acquire N camera frames for inference within a short period of time. If the N acquired frames are similar to each other, using one of the N frames for all the concurrent processes would not change their inference

results, which leads to energy waste for acquiring more camera frames than necessary. Second, **performance-oriented model inference execution.** A computer-vision process on an AMR continuously performs frame acquisition and model inference without waiting time, i.e., without delaying the next frame acquisition and inference time. Although this execution mode ensures high performance (e.g., safety, accuracy), it also prevents the computing hardware (e.g., CPU, GPU) to reach deep sleep states for energy savings. Third, **uncoordinated execution of concurrent jobs.** Multiple executing inference processes executing concurrently, even with a per-process optimized waiting time, may still prevent the computing hardware from reaching the deep-sleep state. In fact, most computing systems can reach the deep sleep state only after a certain amount of time has passed. Thus, uncoordinated waiting times may cause the computing hardware to never reach the necessary contiguous idle time to enter the sleep state.

In order to address the above described inefficiencies, we propose E2M, a generalized energy-efficient middleware for autonomous mobile robots. E2M consists of four major components: sensor buffer, performance analyzer, energy saver, and coordinator. The sensor buffer is designed to capture the sensor data (e.g., camera frames). It allows to coordinate the concurrent access to sensor data and reduce the total amount of data collected for energy savings. The energy saver profiles the energy consumption of the executing processes based on their waiting times. The performance analyzer profiles the performance of each running process based on a predefined per-process metric and desired target. The coordinator collects the information about energy consumption and performance analysis to find the best waiting times for each process that maximize the contiguous idle time of the computing hardware for energy savings while ensuring good performance for each process. In practice, the coordinator controls the waiting time by controlling the feed time of sensor data to each process.

In summary, this paper makes the following three contributions:

- We analyze the computational energy consumption of autonomous mobile robots and find three main sources of inefficiency: uncoordinated access to sensor data, performance-oriented model inference execution, and uncoordinated execution of concurrent jobs.
- We propose an Energy-Efficient Middleware (E2M) for autonomous mobile robots to fix the three inefficiencies. E2M coordinates the access of processes to sensor data and coordinates the execution of the processes to maximize energy savings while ensuring good performance.
- We develop a prototype of E2M on a real-world AMR and test it on a real scenario. Our experimental results show that E2M leads to 24% energy savings for the computing platform, which translates into an extra 11.5% of battery time and 14 extra minutes of robot runtime, with a performance degradation lower than 7.9% for safety and 1.84% for accuracy.

The rest of the paper is organized as follows. Section 2 studies the related work. Section 3 describes our analysis of the AMR's energy consumption. Sections 4 and 5 describe the design of E2M and its implementation, respectively. Section 6 evaluates the proposed solution. Section 7 proposes a discussion on the limitations and future work of E2M. Section 8 concludes the paper.

2 RELATED WORK

Autonomous mobile robots can autonomously execute a large variety of jobs with little to none human intervention. On the other hand, one major limitation of AMRs is their limited battery life, which leads them to often interrupt the executing job and reach the nearest available charging station. In order to increase the battery life of AMRs, a variety of approaches have been proposed by related work.

Previous studies have proposed to find the most energy-efficient path for robots to move to a certain location or to cover a large area [2, 3, 5, 8, 15, 45, 47]. Other solutions coordinate various AMRs to optimize their charging scheduling [6, 20, 21, 23, 24, 32, 35–37]. Most of the above studies focus on the energy consumption of the robots due to the mechanical parts. Different from all the above solutions, in this paper we focus on the computational energy consumption, which can account for the 33% of the total energy consumption.

The energy consumption of robots is mainly influenced by the type of task allocated to its computational unit. To this end, a lot of work has been done to find the best task allocation strategy for AMRs [9, 11, 19, 27, 31, 34, 43]. For example, some approaches attempt to move tasks across robots to save energy considering the distance from the target area or the energy allowance [41, 48]. Other solutions propose to ensure the continuous coverage of a single task (e.g., multiple drones to follow a car) by moving the task across robots in relation to their energy budget [7]. However, none of the above solutions have considered how to minimize the task execution energy consumption on the robots.

A large portion of jobs executed by robots are related to computer vision, which exploit cameras and machine learning models to make the robots more intelligent and autonomous. While some related studies focus mainly on performance [28, 29, 42] without considering the energy consumption, other solutions focus on pruning and compression techniques to make the trained model more energy efficient [38, 46]. However, none of these studies focus on how those applications are executed on the computing hardware of the robots. Note that we still need to manage the computing resources energy consumption, even if every application running on the multi-purpose AMR is pruned. This is because we want to ensure a desired level of performance for each application running on the AMR. However, when multiple applications run concurrently in a shared environment, they are likely to slow down each other [12, 13, 18]. As a result, the amount of computing resources utilized may increase, which may lead to increased energy consumption. E2M is proposed to address this problem and ensure the desired applications performance while minimizing the AMR's computing resources energy consumption. In particular, we find that it is possible to improve their energy efficiency by introducing a short waiting time in each application, which trades off performance for lower energy consumption. In addition, we study how to coordinate the waiting times of concurrently executing applications to minimize the energy consumption due to capturing camera frames and running model inferences. To the best of our knowledge, this is the first work to study and improve the energy efficiency of executing computer-vision applications on AMRs.

3 POWER ANALYSIS OF AN AMR

The deep learning based approach has been widely used in autonomous driving applications. However, currently most of the proposed methods are performance driven without considering the energy consumption. In this section, we first describe our experimental setup used to execute the experiments. Second, we analyze the power breakdown of a real AMR platform. Third, we analyze the power breakdown of computer vision processes to identify the highest sources of energy consumption. Finally, we analyze the effect of the waiting time of processes on the computing power dissipation.

3.1 Experimental Setup

In order to characterize the energy consumption of autonomous mobile robots, we analyze an indoor autonomous mobile robot called HydraOne [44]. The HydraOne platform is shown in Figure 1(b). Different from many heavy-weight AMRs used for one specific application (e.g. moving heavy objects), we consider multi-purpose AMRs, which can run various computer-vision applications concurrently. The user can decide which applications to execute at any time. For example, multi-purpose AMRs can be used in retail stores to help managers execute applications such as understanding out-of-stock items, guaranteeing price integrity, confirming product showcases, and identifying hazardous conditions [1]. Thus, multi-purpose AMRs do not necessarily need to be heavy-weight. The HydraOne configuration is a representative design of such robots with multiple concurrent computer vision applications running on it. In particular, HydraOne is a full-stack research and education platform and includes mechanical components, vision sensors, computing hardware, and communication system. All resources on HydraOne are managed by the Robot Operating System (ROS). Figure 1(a) shows the hardware design of HydraOne. Two leopard cameras [26] and an RPLiDAR [39] are connected to the computing platform via USB cable. RPLiDAR is a 2D laser scanner that provides 360 degree laser range scanning. The results of a set of data points in space is called *point cloud*. An Nvidia Jetson TX2 board is used as the computing platform [33]. One Arduino Mega 2560 board with two motor driver boards are used to control HydraOne. Two 3S Lipo battery is used to power the whole system: one for the computing platform; the other for the wheels. The capacity of each 3S Lipo battery is 5000mAh.

On top of HydraOne, we implement a deep learning based end-to-end free space detection application and an object detection application. Unfortunately, existing navigation applications cannot be used in our experiments because they are not designed and trained in our environment. As a result, we designed and trained a Convolutional Neural Network (CNN) called HydraNet to achieve end-to-end free space detection, where the input is the frame from camera and the output is the controlling command, i.e., linear and angular speed to the robot. The training dataset, which contains the image frames labelled with control messages, is obtained by a human remotely controlling HydraOne. There are five convolutional layers and four Fully Connected Network (FCN) layers in HydraNet. The convolutional layers are designed to perform feature extraction. The first three convolutional layers have a 5×5 kernel and a 2×2 stride. The last two convolutional layers have a non-strided convolution with a 3×3 kernel. The filters of these five layers are 24, 36, 48, 64, and 64, respectively. Four FCN layers are designed as the decision maker for the driving and lead to two output values of linear and angular speed. The number of cells in each FCN layer are 512, 100, 50, and 10, respectively. Based on the experience of industry practice and a series of experiments, we choose Rectified Linear Unit (ReLU) as the activation function for all nine layers. The RPLidar is used in this context to collect data about objects distances. For object detection, we choose the combination of MobileNet and Single Shot MultiBox Detector (SSD) [30]. Through replacing the lightweight depthwise separable convolution layer with standard convolution layer to reduce the number of computations, MobileNet becomes more suitable for the resource-constrained AMRs platform [17]. Furthermore, SSD is a widely used deep learning model for objection detection.

The power dissipation of the computing platform and sensors is measured using Watts Up Pro Electricity Consumption Meter [16], which records both the current and real power every second. The power dissipation of the wheels is measured using the Lipo battery charger, which records the energy consumption. The error of power dissipation is less than 1 percent and it is ignorable. The reason why we use two different meters is that we have two batteries installed on HydraOne, one powering the locomotive mechanisms (e.g., wheels) and the other one powering sensors and computing resources. We run HydraOne with free space detection and object detection for 10 minutes and use the charger and the Watts Up Pro to measure the energy consumption. Then we calculate the average power dissipation based on the energy consumption and running time. When the batteries are fully charged, HydraOne platforms can run with HydraNet and MobileNet-SSD for approximately two hours.

3.2 AMR Power Breakdown

Figure 2 shows the power dissipation of the entire HydraOne platform. We run HydraNet and MobileNet-SSD on HydraOne for 10 minutes. The total power dissipation of HydraOne is 39.1W. From Figure 2, we can see that the locomotion of HydraOne (*Wheels* in the figure) consume over half of the total power dissipation, the power dissipation of computation is 33%, and that of the sensors is 11%. Out of the 33% of computational power dissipation, the model inference of HydraNet and MobileNet-SSD consume 10% and 12% of the total power dissipation of the robot, respectively. The remaining 10% indicated with *other* in the figure includes the power dissipation of the operating system and sensor drivers.

As we can observe from this analysis, each implemented application may further increase the total computational power dissipation. Actually, in many cases, AMRs may have to run more than two applications. For example, a surveillance AMR may implement, other than the free space detection and object recognition application, additional applications for face recognition (to timely detect the identity of intruders), self-diagnostic, and other third-party applications deemed necessary by the user. This fact leads to the following observation:

Observation 1: *The AMR's computational power dissipation can highly influence its autonomy. It is thus necessary to optimize the computing system of AMRs for high energy efficiency.*

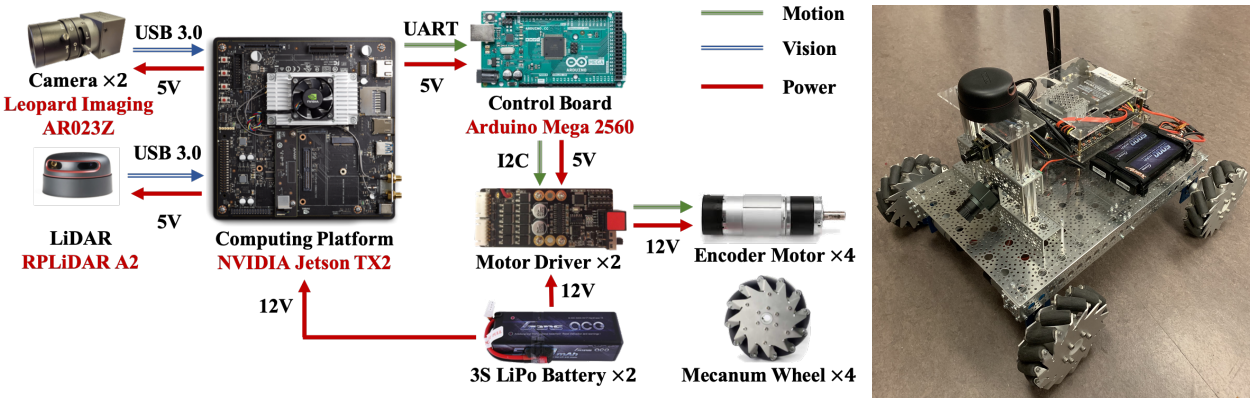


Figure 1: (a) The hardware design of HydraOne platform. (b) The HydraOne platform.

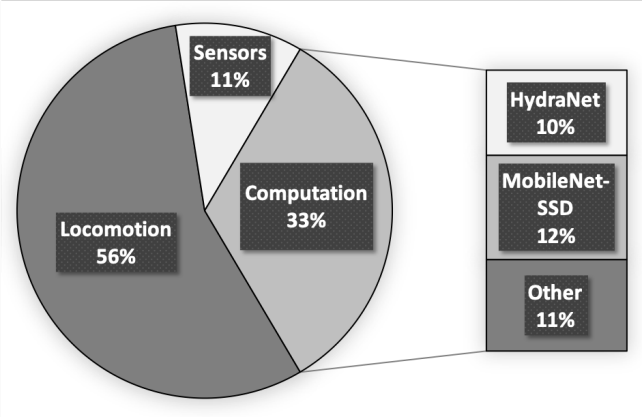


Figure 2: Power dissipation breakdown of an AMR.

Table 1: Power Dissipation breakdown of HydraNet and MobileNet-SSD.

Power Dissipation (W)	HydraNet	MobileNet-SSD
Capturing Frame	2.2	2.2
Model Inference	0.8	2.5
Publish Results	0.9	0.1

3.3 Computer Vision Power Breakdown

We now analyze how the power is used by each application implemented on HydraOne. These applications use deep learning to detect free space and objects. In general, each application executes a process for the detection that works in three steps. In the first step, the process captures a frame from the camera. In the second step, the process executes model inference. In the last step, the process publishes/shows the results. The publisher/subscriber mechanism is used for data communication in ROS. We believe that these three steps are general enough to characterize most of the other applications and processes not tested in this paper. In order to understand the energy consumption of each running process, we measure the power dissipation during each one of the above described three steps for HydraNet and MobileNet-SSD. When no waiting time of the application is applied, the power breakdown results are shown in Table 1.

From the power breakdown analysis in Table 1, we can see that capturing frames from camera consumes 2.2W for both HydraNet and MobileNet-SSD. The power cost of model loading and inference is 0.8W for HydraNet and 2.5W for MobileNet-SSD. The reason why MobileNet-SSD consumes more energy than HydraNet is that MobileNet-SSD has more layers and neural cells than HydraNet. After model inference, the results of HydraNet are published to another ROS node, which executes the control message on HydraOne. For object detection, the result will be shown to the user. The power cost for publish/show results is 0.9W for HydraNet and 0.1W for MobileNet-SSD. The difference is owing to the overhead of ROS's publisher/subscriber mechanism. Thus, on average across different processes, capturing frames costs 2.2W, running the model inference costs 1.65W, and publishing/showing the results costs 0.5W. An important observation must be made about the application access to camera frames. Currently, applications access the camera directly and exclusively, which means that when an application is accessing the camera, other applications can either wait to access it or access another camera (if available) to avoid performance degradation. Thus, the total energy consumption of two applications accessing the camera frames could potentially be halved by accessing the camera once and sharing the frame among the two requesting applications. Note that the energy savings with this method increase with the number of concurrent applications. As it can be observed in Table 1, the frame capturing and model inference consumes the 88% of the total process power dissipation. This leads to the following observation:

Observation 2: *High power dissipation in AMR processes is mainly due to capturing camera frames and executing the model inference. Thus, to improve the energy efficiency of AMRs, we need to focus on these two steps.*

3.4 Waiting Time vs Power Dissipation

Given the above observations, we need to find a way to improve the energy efficiency of capturing camera frames and executing model inference for AMR processes. To do so, we explore the possibility to introduce a *waiting time* between two consecutive inferences of computer vision processes. The waiting time is defined as the amount of time between the end of a model inference and the start of the next one. As mentioned in Section 1, currently these processes execute inferences continuously, which leads to highest performance but also to high energy consumption. By introducing a waiting time, we are delaying the execution of the next inference to trade off performance for energy savings. Note that the waiting time, other than reducing the energy consumption due to the inference, reduces also the number of camera frames captured. Thus, by manipulating the waiting time we can improve the energy efficiency of the two steps in computer vision processes that consume the highest amount of energy. Here, we use the term performance to indicate in general a specific metric associated with each process. For example, the performance metric for the free space detection application is safety, i.e., distance of the robot from obstacles, while the performance metric for the object detection is accuracy, i.e., how many objects are actually recognized. It is of primary importance to meet a desired performance target while reducing the energy consumption.

To test the effect of the waiting time on energy consumption, we run HydraNet model inference and MobileNet-SSD model inference with different waiting time on the HydraOne platform and measure the power dissipation due to the computing system. We conduct the experiments with waiting time from 0 to 0.5 seconds. The results are shown in Figures 3(a) and 3(b). Introducing a waiting time in HydraNet could degrade the safety of HydraOne, i.e., the average distance of HydraOne from the surrounding objects decreases with an increasing waiting time. In relation with Figure 3(a), Figure 3(c) shows the distance of HydraOne from the surrounding objects for different waiting times. As the figures show, the waiting time can decrease the power dissipation of the computing platform by up to 40% for HydraNet and 35% for MobileNet-SSD with less than 0.15m increase of distance to objects. The power consumption of HydraOne is 7.5W when no inference is executed. Adding the HydraNet inference with zero waiting time, i.e., as the *baseline* of reference, increases the power dissipation of the platform to 12.7W on average, which corresponds to a 41% power increase (similar for MobilNet-SSD). Introducing the waiting time can help reduce the power dissipation due to the inference. In particular, by introducing a 0.1s waiting time in HydraNet (i.e., capture frame and run inference every 100ms), the power dissipation decreases to 11.4W, which is 10% lower than no-wait. On the other hand, further increasing the waiting time leads to smaller increases in power savings and reaches an average of 10.5W for a 0.5s waiting time, which corresponds to a 17% power reduction. The reason for this marginal increases in power savings for an increased waiting time is that the average power consumption of the computing platform when waiting time is applied saturates to a minimum power consumption for longer waiting times. In addition, the performance degradation is acceptable when introducing small waiting times. From Figure 3(c), we can observe that the performance of 0.1s waiting time is similar to that of the baseline. Even for 0.5s waiting time, the increase of distance to objects is less than 0.15 meters, which is still acceptable. Thus, high power reductions and small performance degradation can be obtained using small waiting times. Note that MobileNet-SSD shows similar behavior but has less sensitivity to increasing waiting time because MobileNet-SSD has a much longer model inference time than HydraNet. These experimental results lead us to the following observation:

Observation 3. *The relation between the waiting time and the power dissipation is non-linear and shows high power reductions for small waiting times. Thus, it is possible to trade off a small performance degradation for high energy savings.*

4 ENERGY EFFICIENT MIDDLEWARE

The goal of the Energy Efficient Middleware (E2M) is to provide a general software solution to make the computer vision based applications on autonomous mobile robots more energy efficient. Although there are several different computer-vision applications, for simplicity here we describe the design of E2M based on two specific applications that we have fully implemented on our testbed platform HydraOne, i.e., object detection, which recognizes objects from camera frames, and end-to-end free space detection, which directly commands the speed and direction of the robot based on camera frame analysis. However, the design of E2M is general enough so that it can be easily used for any type of computer-vision applications. The design presented in the next sections is based on introducing waiting time and optimizing the system overhead for frame capturing and model loading. First, we present the overview of E2M. Then, we discuss the details of each application within the E2M system.

4.1 Overview

Figure 4 shows the overview of E2M, which consists of four components: the sensor buffer, the performance analyzer, the energy saver, and the coordinator. E2M locates in the middle between the lower level hardware (i.e., sensors and wheels) and the applications (e.g., free space detection and object detection).

The sensor buffer communicates with the sensors including camera and RPLiDAR to get image frames and point cloud data. Also, the linear and angular speed can be read from the wheels' ROS node. A node is an executable that uses ROS to communicate with other nodes [1]. Wheel's ROS node is used to send control messages to the motor's driver. The performance analyzer subscribes to the sensor data from sensor buffer and conducts the model inference of each application. In addition, it measures the performance metric and calculates the maximum waiting time for each running application. Because each application has a specific objective, each application has a specific performance metric. For example, the performance of free space detection is determined by the safety of the robot. For object detection, the performance is determined by the accuracy in recognizing objects. Because each application has its own metric, in order to include a new application in E2M, the user (e.g., developer) just needs to input the performance metric, which will then be measured by the performance analyzer for E2M decisions. The energy saver automatically estimates the energy consumption of each application for various waiting times. All the waiting time,

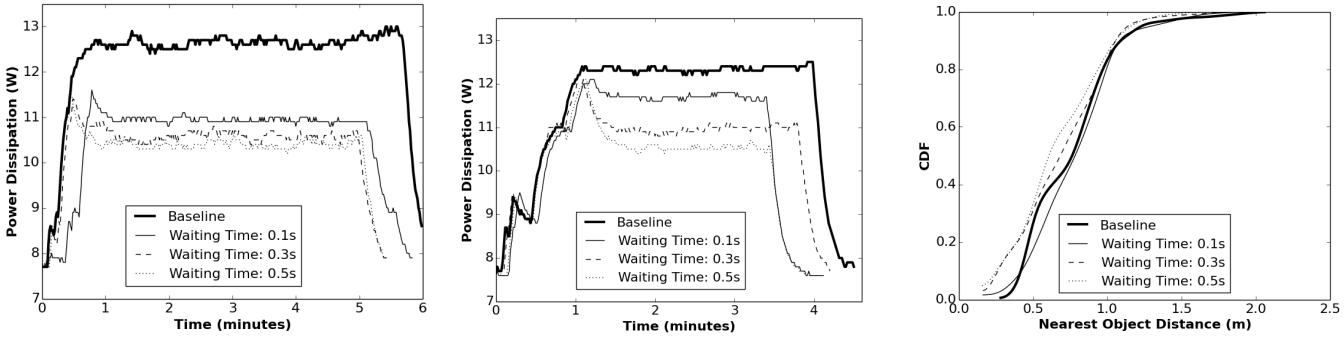


Figure 3: Effect of waiting time on (a) HydraNet and (b) MobileNet-SSD. (c) The distance to the nearest object CDF.

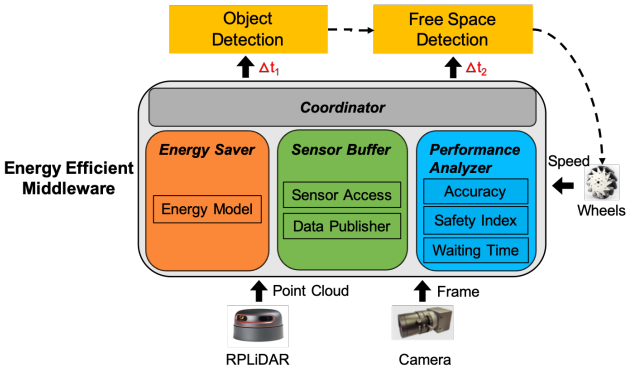


Figure 4: E2M high-level overview.

energy model, and performance analyzer results are sent to the coordinator, which determines how multiple applications could work together to maximize the energy saving. The speed of the AMR is an important input to E2M and it affects not only the performance requirements of each application but also the energy saving of E2M.

4.2 Sensor Buffer

From Section 3, we know that uncoordinated access to sensor data is one of the inefficiency sources of energy of AMRs. In fact, multiple computer vision based applications running simultaneously can share the sensor but have exclusive access to it to ensure reliability. The reason for this inefficiency is that the sensor driver gives exclusive access to the application reading the sensor. This means that multiple applications accessing the sensor within a short period of time may read similar data multiple times, thus wasting sensor's driver and data capturing energy. Therefore, E2M implements a sensor buffer application to share the video frames among multiple applications to reduce the energy consumption.

The sharing of the video frames across concurrently executing applications is based on ROS's Publisher/Subscriber mechanism. The design is to capture the sensor data from camera and RPLiDAR and share the data with other ROS nodes to process the data. The size of the sensor buffer is statically defined and ROS manages to drop the extra data.

Each application subscribes to the data it requires from the sensor buffer. During the waiting time periods where no application need data from a sensor, the sensor data will still be captured and published by sensor buffer, but the data is not subscribed by other ROS nodes. There are two reasons for this design. First, although it is possible to stop the running sensor buffer, the sensor device does not support the operation of shutdown by receiving a command from the computing platform. Second, the waiting time is on the milliseconds level while the time it takes to restart the sensor node and publish data out is on the seconds level. If the sensor is turned off, the data will be lost before it restarts. Therefore, the sensor buffer just keeps running in the design of E2M.

4.3 Performance Analyzer

E2M is designed to save energy with guaranteed performance. Thus, how to quantify the performance becomes an essential problem. In general, the performance metric must be defined based on the application. Because free space detection determines not only the direction but also the speed of the robot, the performance is determined by the safety of the robot. For object detection, the performance is determined by the detection accuracy. Thus, we define the cumulative accuracy for object detection to evaluate its performance. One problem is how to compare different performance metrics to make decisions about the coordinated waiting time of applications. In order to correctly compare different metrics, all the performance metrics are normalized using their desired value. Thus, for any application, a good performance is achieved with a normalized performance value near one. Next, we define two metrics to quantify the performance of the two example applications implemented on HydraOne, i.e., the safety index and the cumulative accuracy.

Safety Index. In this paper, we assume that if the robot doesn't crash into any object, then the control of the robot is safe. Based on this assumption, we propose a safety index that aims at keeping HydraOne from collision. As Figure 5 shows, there are two cases of HydraOne's safety index computation. The linear (V) and angular (α) speed of the robot use polar coordinates with origin in the front of the robot. For cases 1 and 2, the idea is to calculate the safety

index as the subtraction between the front nearest object distance and the braking distance. The robot's speed is decomposed into the front and the right directions of the robot. The AMR distinguishes case 1 with case 2 based on the point cloud data from RPLiDAR. The idea is to compare the change of distance around the degree with the shortest distance: if the shortest distance value is the trough then E2M uses case 1, otherwise it uses case 2.

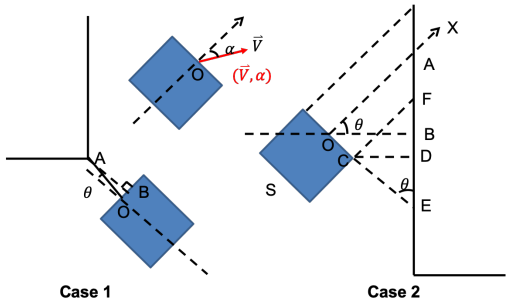


Figure 5: Two cases of safety index.

Case 1: The robot heads to a corner of the wall. In this case, the difference between the shortest distance to obstacles and braking distance gives the safety index for case 1, which is expressed as follows:

$$S_1(v(t), a(t)) = \text{a(t)}\cos\theta(t) - \left[v(t)\cos\alpha(t)t_0 + \frac{(v(t)\cos\alpha(t))^2}{2\mu g}\right] \quad (1)$$

In Equation 1, t is the time and $v(t)$ is the speed of the robot at time t. $S_1(v(t), a(t))$ represents the safety index of case 1 given the speed $v(t)$ and the shortest distance $a(t)$. The shortest distance OA is represented as $a(t)$ and the degree between OA and the front direction is denoted as $\theta(t)$. t_0 represents the reaction time and μ is the static fraction coefficient. μ reflects the relationship of static fraction with robot's weight and it's determined by the materials of wheels and ground.

Case 2: The robot heads toward a wall. In this case, the shortest distance that can be read from RPLiDAR's scan message is OB. Because the robot can only go forward, when reading the message from RPLiDAR only those messages whose degree is within $-\pi/2$ to $\pi/2$ are used to find the nearest point. Both OB and the robot's speed are decomposed in the front direction and right direction of the robot. Then we can calculate the difference between shortest distance to obstacles and braking distance for each direction. Then the lower difference between the two directions is chosen as the safety index. Equations 2, 3, and 4 show how to calculate safety index for case 2.

$$d_x(t) = \frac{a(t)}{\cos\theta(t)} - \frac{s}{2}\tan\theta(t) - \left(v(t)\cos\alpha(t)t_0 + \frac{(v(t)\cos\alpha(t))^2}{2\mu g}\right) \quad (2)$$

$$d_y(t) = \frac{a(t)}{\sin\theta(t)} - \frac{s}{2} - \left(v(t)\sin\alpha(t)t_0 + \frac{(v(t)\sin\alpha(t))^2}{2\mu g}\right) \quad (3)$$

$$S_2(v(t), a(t)) = \min\{d_x(t), d_y(t)\} \quad (4)$$

The speed $v(t)$ is decomposed into x axis(front) and y axis(right). $d_x(t)$ represents the shortest distance in the x axis at time t. $d_y(t)$ represents the shortest distance in the y axis at time t.

Cumulative Accuracy. For object detection, we use accuracy as performance metric. Currently, most related work solutions evaluate the performance of object detection algorithms by utilizing Intersection of Union(IoU) and mean Average Precision(mAP) [14]. However, for a trained object detection model (e.g., MobileNet-SSD in HydraOne), the detection accuracy for a specific frame does not change in different executions because we do not change the trained model. Assuming the robot has 0.1 seconds waiting time, if we compare the object detection results with waiting time and without waiting time, the average IoU and mAP will be the same because the DNN model is the same. However, objects during the waiting time are not analyzed by the model. As a result, the baseline that continuously runs without waiting time will have the highest number of objects recognized during a certain activity period. Therefore, we design the cumulative accuracy (CA) to measure how many objects are analyzed and recognized during a certain active period and evaluate the object detection performance:

$$CA = \sum_{i=1}^{N} \max\{a(i,j), j \in [1, M_j]\} \quad (5)$$

where N is the number of times the model inference has been executed. For the $i-th$ model inference, $a(i,j)$ represents the output of the model inference, which is a vector consisting of M_j elements, and each element indicates the probability of being recognized as a particular object. Thus, $\max\{a(i,j), j \in [1, M_j]\}$ selects the recognized object with the highest probability for each model inference execution. As a result, CA is the sum of all the highest possibilities over N times of model inference.

Metric Normalization. The performance metrics of different applications and their ranges can be different. The goal of normalization is to make E2M a general support for all computer vision based applications in AMRs and to easily compare the performance of different applications.

In order for the coordinator to make a final decision on the waiting time of the applications, the coordinator needs to consider the performance of each application (see Section 4.5). Therefore, the performance metrics should be normalized. For safety index, first we collect the human driving dataset by using the joy stick to control HydraOne with no crash. The lower the safety index is, the higher is the possibility for a crash to happen. Then the normalized safety index can be calculated using the desired safety index, which is determined as the largest safety index when the robot is controlled by a human being. For CA, we also use desired CA value to divide the real-time CA value. The desired CA values is defined as the total number of objects through the detection process.

$$N_S = \frac{S(v(t), a(t))}{S_{des}} \quad (6)$$

$$N_{CA} = \frac{CA(v(t), b(t))}{CA_{des}} \quad (7)$$

In Equations 6 and 7, N_S and N_{CA} represent normalized safety index and normalized cumulative accuracy respectively. The higher

N_S is, the more safe is the AMR. The higher N_{CA} is, the better the AMR recognizes objects. Specifically, S_{des} and CA_{des} represent the desired safety index and cumulative accuracy, respectively. As a result, the performance analyzer also calculates the per-application waiting time. The maximum waiting time for HydraNet and MobileNet-SSD can be calculated based on the distance and speed as follows:

HydraNet waiting time

$$W_{AD}(v(t), a(t)) = \frac{S(v(t), a(t))}{v(t)} - t_{AD} \tag{8}$$

MobileNet-SSD waiting time

$$W_{OD}(v(t), b(t)) = \frac{\min\{b(t)\}}{v(t)} - t_{OD} \tag{9}$$

In Equations 8 and 9, $W_{AD}(v(t), a(t))$ and $W_{OD}(v(t), b(t))$ represent the maximum waiting time for HydraNet and MobileNet-SSD respectively given the speed $v(t)$ and distance information $a(t)$ and $b(t)$. The higher $W_{AD}(v(t), a(t))$ and $W_{OD}(v(t), b(t))$ are, the more energy can be saved. $a(t)$ represents the shortest distance to the RPLiDAR and t_{AD} is the average inference time for HydraNet. $b(t)$ is the distance to objects and t_{OD} is the average inference time for MobileNet-SSD. The distance to the objects are predicted based on the RPLiDAR's *scan* message and the object's location in the image.

4.4 Energy Saver

Energy saver is designed to determine the total energy consumption of the system when the waiting time of each application changes. The results from the energy saver are directly fed into the coordinator. Equations 10, 11, and 12 show the energy model before applying E2M:

$$E_{Baseline} = E_{other} + E_{AD} + E_{OD} \tag{10}$$

$$E_{AD} = E_{AD-capture} + E_{AD-inference} + E_{AD-publish} \tag{11}$$

$$E_{OD} = E_{OD-capture} + E_{OD-inference} + E_{OD-show} \tag{12}$$

$E_{Baseline}$ is the total energy consumption when E2M is not applied. $E_{Baseline}$ consists of three components: E_{AD}, which is the energy consumption of running end-to-end free space detection application, i.e., HydraNet; E_{OD}, which stands for the energy consumed by running object detection application, i.e., MobileNet-SSD; E_{other}, which represents the energy consumption of other parts, including sensors powering and wheels rotation. For each running application, HydraNet and MobileNet-SSD, we further break the energy consumption into sensor data capturing $E_{AD-capture}$ or $E_{OD-capture}$, model inference $E_{AD-inference}$ or $E_{OD-inference}$, and publishing commands/showing results $E_{AD-publish}$ or $E_{OD-show}$.

After applying E2M, the energy model can be represented as:

$$\begin{aligned} E_{E2M} = {} & E_{other} + E_{sensor-buffer} + E_{E2M-overhead} \\ & + \frac{t_{AD} - \sum W_{AD}}{t_{AD}}(E_{AD-inference} + E_{AD-publish}) \\ & + \frac{t_{OD} - \sum W_{OD}}{t_{OD}}(E_{OD-inference} + E_{OD-publish}) \end{aligned} \tag{13}$$

From Equation 13, we can see that sensor buffer is introduced to capture sensor data instead of letting each application access the sensor directly. E_{E2M} represents the total energy consumption after applying E2M. $E_{sensor-buffer}$ represents the energy consumption of capturing and sharing sensor data. The length of waiting time of each application also affects the energy consumption of model inference and output. In addition, there are computation overhead in implementing the system: safety index, cumulative accuracy, and publisher/subscriber mechanism in ROS. Here we use $E_{E2M-overhead}$ to represent the total energy overhead.

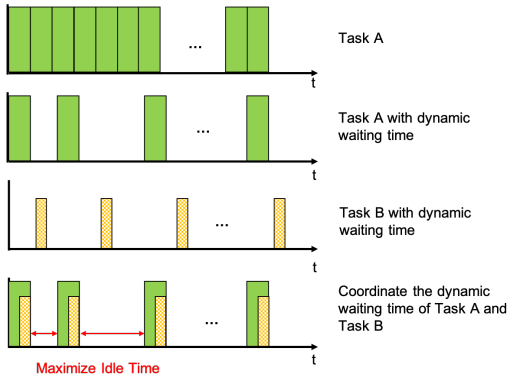


Figure 6: The design of how E2M coordinates multiple applications.

4.5 Coordinator

As anticipated in Section 3, the uncoordinated execution of concurrent jobs is another source for inefficiency on AMRs. The coordinator is proposed to coordinate multiple applications so that the overall energy consumption is minimized. Our design is shown in Figure 6. We can see that there are two applications: application A and application B. Based on the energy and waiting time model we have discussed above, the maximum waiting time and energy saving for each application can be calculated. However, the problem is how to coordinate them to minimize the energy consumption while ensuring good performance. Our idea is to coordinate the waiting time and inference time to maximize the idle time. Indeed, increasing the idle time of computing resources helps the computing hardware to reach the deep-sleep state more often for energy savings.

In E2M, we assume the presence of a safety-related application simply because autonomous mobile robots need at least one navigation application like HydraNet to be executed. Navigation applications are often related to a safety index performance metric. As a result, the main role of the coordinator is how to coordinate the safety-related application with the other applications to minimize the energy consumption. Therefore, we enforce a restriction on their calculated waiting time. The maximum waiting time of the other applications will not exceed that of the navigation application. For example, if the computed waiting time of MobileNet-SSD is longer than the waiting time of HydraNet, we assign the waiting time of HydraNet to MobileNet-SSD. The reason for implementing this restriction is that the execution of HydraNet's inference changes the speed and direction of HydraOne, which indirectly affects also the performance and thus the waiting time of the other

applications. As a result, the coordinator optimization problem is defined as follows:

$$\begin{aligned} \textbf{min} \quad & E_{E2M} \\ \textbf{s.t.} \quad & 1 - \varepsilon_k < N_k < 1 + \varepsilon_k \\ & 0 \leqslant W_k \leqslant \min(W_k^{max}, W_S^{max}) \quad \forall k \in [1, A] \end{aligned} \tag{14}$$

For every coordinator activation, the objective is to minimize the energy consumption of the computing platform and the decision variables are the waiting times for each application for the current coordination period. Assuming there are A applications, for each application k there are three restrictions: the first is that the normalized performance metric of the application k should be close to the desired value, i.e., one, where ε_k is defined by the user and represents the acceptable error from the desired value; the second is that the waiting time for application k is positive and it is lower than the minimum between the maximum desired waiting time W_S^{max} of the navigation application S and the maximum desired waiting time W_k^{max} of the k^{th} non-navigation application (e.g., as calculated for HydraNet and MobileNet-SSD in Equation 8 and 9, respectively).

When there is only one application running (i.e., the navigation application), the output waiting time of the sensor buffer is directly applied without the need of publishing/sharing results. Because the coordinator is needed when multiple applications concurrently execute, here we cover the case when there are more than one applications running in parallel. In addition, because we don't predict performance analysis and energy savings over future periods, the coordinator's work is to find the optimal waiting time with guaranteed performance for the current coordination cycle. The optimization process consists of five main steps:

Step 1: Wait for a new inference of the navigation application. When the navigation application finishes the current inference, its waiting time W_S is calculated. To minimize the energy consumption, by default the waiting times of all the applications are set to their max value (i.e., $N_k = 1\ \forall k \in [1, A]$).

Step 2: For the non-navigation applications with waiting time W_k higher than W_S, assign the waiting time of navigation application to these application ($\forall k \in [1, A]$, s.t. $W_k > W_S$, then $W_k = W_S$). The reason is that the navigation application controls the speed and direction of the robot. If the other non-navigation applications wait more time than the navigation application, then the speed and direction of the robot changes during their waiting time, which may degrade their performance.

Step 3: For the non-navigation applications that have a shorter waiting time than the navigation application ($\exists k \in [1, A]$ *except* S, s.t. $W_k < W_S$), get the minimum waiting time W^{min} among these non-navigation applications;

Step 4: After W^{min} has passed, execute all the model inference of non-navigation application whose waiting time is shorter than W_S.

Step 5: After all the applications have finished their model inference, check if there is still enough time to wait for the new minimum waiting time plus the model inference time. If yes, go to Step 3. If not, go to Step 1.

Note that we use this heuristic algorithm to solve the above coordinator optimization problem. We plan to improve the design of the coordinator algorithm in our future work to provide theoretical guarantees of optimality.

5 IMPLEMENTATION

The implementation of E2M is conducted on HydraOne platform. The HydraOne platform is shown in Figure 1(b). HydraOne is an indoor autonomous mobile robot embedded with two leopard cameras and an RPLiDAR. The leopard camera has 1928H x 1088V active pixels and a frame rate of 30fps. The RPLiDAR's accurate range is from 0.2 meters to 12 meters, which is enough for indoor scenarios. Each sample acquired by the RPLiDAR is composed of 8000 distance points around 360 degree view. The computing platform is an NVIDIA Jetson Tegra X2 board that has an NVIDIA $Pascal^{TM}$ Architecture GPU, 2 Denver 64-bit CPUs with Quad-Core A57 Complex, and an 8 GB L128 bit DDR4 Memory. The cameras and RPLiDAR are connected to the Jetson TX2 board via USB 3.0 port.

The ROS framework design of E2M is shown in Figure 7. An ROS node (i.e., the nodes in the figure) is a process to perform a certain computation while ROS topics (i.e., the arrows in the figure) are named buses for ROS nodes to exchange messages [1]. In E2M, we implement seven ROS nodes to access the sensor data and process the data. Six ROS topics are implemented to exchange messages including images, point clouds, control commands, and other customized messages.

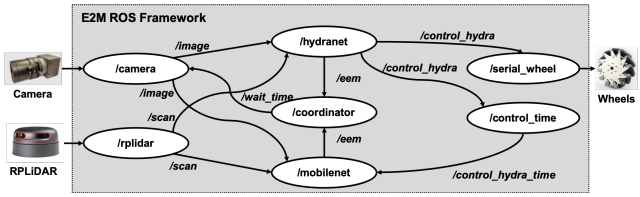


Figure 7: The ROS framework of E2M.

5.1 ROS Nodes

Each ROS node runs as a process to perform computation. Among the seven ROS nodes, */camera* and */rplidar* are used to access the sensors to capture and publish the data. Thus, the sensor buffer is implemented within these two nodes. The performance analyzer and energy saver are implemented in ROS node */hydranet* and */mobilenet* accordingly. The ROS node */hydranet* also executes the model inference of HydraNet for free space detection and calculates the safety index as well as waiting time for HydraNet. The ROS node */mobilenet* executes the model inference of MobileNet-SSD for object detection and measures the cumulative accuracy and waiting time. The ROS node */serial_wheel* is designed to get the control message */control_hydra* from */hydranet* and publishes them to the Arduino board that controls the motor drivers. */control_hydra* doesn't have timestamp but it is necessary for ROS node */mobilenet* to synchronize the messages, so the ROS node */control_time* is introduced to convert the control message */control_hydra* to a timestamped control message */control_hydra_time*. The */coordinator* subscribes to

messages including waiting time, performance analysis, and energy saving to calculate the coordinated waiting time then publishes it to */camera*.

5.2 ROS Topics

The ROS topics are defined to exchange messages between ROS nodes. There are six ROS topics in E2M. The summarized description of these six topics are reported in Table 2. The ROS topics */image* and */scan* belong to */sensor_msgs* while */control_hydra* and */control_hydra_time* belong to the */geometry_msgs* library. The header of each ROS topic includes the timestamp of the message. The image pixels are stored in the field "data" of */image*. The distance message is stored in the field *ranges* of */scan*. The ROS topic *control_hydra_time* is generated by the ROS node *control_time*, which subscribers to the *control_hydra* message and adds to it the header with the timestamp for */mobilenet*.

Two customized ROS message */eem* and */wait_time* are defined based on *String* type in */std_msgs*. */eem* is defined for ROS node */hydranet* and */mobilenet* to share some real-time information including waiting time, safety index, and accuracy with ROS node */coordinator*. */wait_time* is defined to send to information of coordinated waiting time to the sensor buffer */image*.

5.3 Message Synchronization

The synchronization issue arises when one application needs to process data from different sensors. In E2M, each ROS node is an individual process that can run anytime, so they can all run in parallel. The ROS topics are used for ROS nodes to share message. However, as the communication delay is a random process, the messages published at the same time may not arrive at the subscribers at the same time. For ROS node that subscribes to multiple ROS topics and processes the messages together, the synchronization of different ROS topics become an essential problem. For example, both ROS nodes */hydranet* and */mobilenet* subscribe to topics */image* and */scan*. For distributed systems, the synchronization is usually based on time. Thus, in order to synchronize multiple ROS topics we first need to attach timestamp to all the ROS topics. Therefore, we add a ROS node */control_time* to transform the control message to a timestamped control message.

When an ROS node subscribes to ROS topics, the node uses a callback mechanism to get the real-time message. When multiple ROS topics are subscribed, the callback function should be called after the synchronization of all the ROS messages. In E2M, the synchronization of multiple ROS topics is implemented based on *message_filters*. We use the *ApproximateTime* policy with buffer length equals to 10 and the slop equals to 0.05s. The buffer is used to store the unsynchronized messages while the slop defines the maximum timestamp difference for synchronization.

6 EVALUATION

In this section, we evaluate our E2M system. First, we describe the testbed configuration used for the experiments. Second, we compare E2M with several baselines to evaluate the computational power dissipation. Third, we evaluate the performance of E2M in terms of safety, accuracy, and latency compared to several baselines. Finally, we describe the resource consumption of the robot with and without E2M by comparing CPU, GPU, and memory activity.

6.1 Testbed Configuration

The system implementation and experiments are all based on the HydraOne platform. According to Figure 2, the energy consumption of sensors on HydraOne occupies the 11% of the total energy consumption. In the experiments, two cameras and one RPLiDAR are connected to the computing platform to implement various baseline cases. The leopard camera consumes 1.3W because of the connection to the Jetson TX2 board via USB port and another 2.2W for capturing frames for applications like HydraNet and MobileNet-SSD. The RPLiDAR consumes 1.9W because of the USB connection and another 0.9W during the RPLiDAR scanning.

In order to correctly determine the waiting time, we first need to profile the model inference time of HydraNet and MobileNet-SSD. Figure 8(a) shows the cumulative distribution function (CDF) of the inference time for the baseline system (i.e., without waiting time) while running HydraNet and MobileNet-SSD. We execute several inferences with the two models and collect the inference execution time for each execution. As the figure shows, 80% of the HydraNet inferences is less than 55ms and 80% if the MobileNet-SSD inferences is less than 320ms. The model inference time of MobileNet-SSD is much higher than HydraNet because MobileNet-SSD has much more convolutions layers than HydraNet. Therefore, we set t_{AD} and t_{OD} as 55ms and 320ms, respectively.

6.2 Power Dissipation

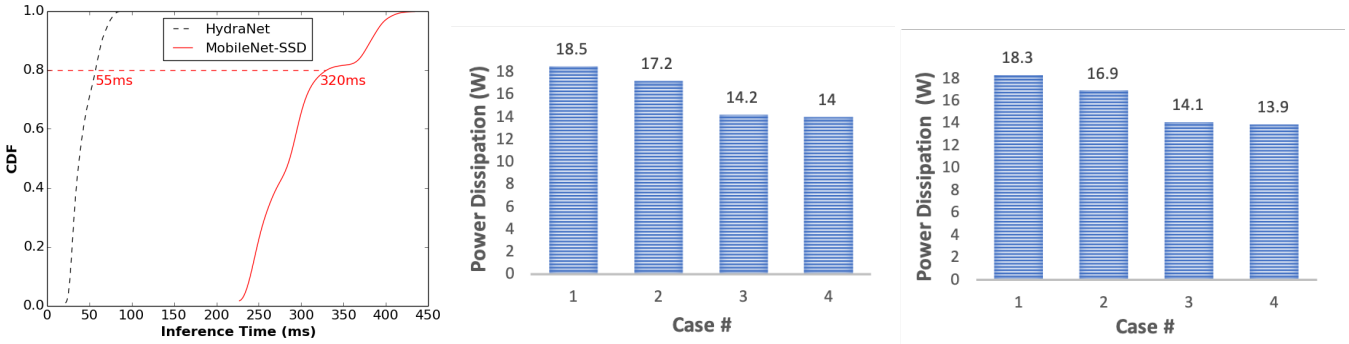
The main objective of E2M is to provide an energy efficient middleware for computer vision applications. Here, we measure the power dissipation of HydraOne's computing platform to study the energy savings introduced by E2M. There are three aspects of energy optimization in E2M: coordinated sensor access, applying of waiting time, and coordination of concurrent running applications. In order to have a breakdown of energy savings for each one of these aspects, we define seven baseline cases. Each baseline introduces one or more of these optimizations.

The description of the seven baseline cases are shown in Table 3. The case 1 is the baseline to cases 2 to 4. The case 1 considers the situation where HydraNet and MobileNet run using two cameras because each application locks the camera to capture the frame and execute model inference periodically. Case 1 also includes the power dissipation of RPLidar driver, which is 1.9W of power dissipation. As the data is not used for any computation, the runtime power dissipation of RPLiDAR will not be included. The case 1 is the general approach to run HydraNet and MobileNet concurrently if we don't have E2M because the camera can be locked by one application. The case 2 shows the power dissipation when HydraNet and MobileNet-SSD are running with the sensor buffer. The introduction of sensor buffer makes it possible to use only one camera for the execution of two applications. The case 3 improves the case 2 by using the waiting time for each individual application. The case 4 executes both applications by improving case 3 with the coordinator, which is the case to show the performance of E2M.

The power dissipation of the above four cases are shown in Figure 8. Specially, Figure 8(b) and Figure 8(c) show the results when

Table 2: ROS topic description in E2M.

Topics	*/image*	*/scan*	*/control_hydra*	*/contro_hydra_time*	*/eem*	*/wait_time*
Library	sensor_msgs	sensor_msgs	geometry_msgs	geometry_msgs	std_msgs	std_msgs
Type	Image	LaserScan	Twist	TwistStamped	String	String
Fields	header, height, width, encoding, data, etc	header, angle_min, angle_max, angle_increment, ranges, intensities, etc	linear, angular	header, linear, angular	header, wait_ad, wait_od, safety, accuracy	header, time



(a) (b) (c)

Figure 8: (a) The inference time CDF of HydraNet and MobileNet-SSD. (b) The power dissipation of E2M in seven cases when the GPU memory utilization ratio is 0.2. (c) The power dissipation of E2M in seven cases when the GPU memory utilization ratio is 0.8.

the GPU memory utilization ratio is 0.2 and 0.8, respectively. As Figure 8(b) shows, comparing case 1, 2, 3, and 4, we can see the introduction of the sensor buffer, the waiting time, and the coordinator saves 1.3W, 3W, and 0.2W of power dissipation, respectively. Although most of the power dissipation is saved by the introduction of the waiting time in each application, we have to note that the energy savings introduced by the coordinator are also dependent on the number of running applications. Because so far we have implemented only two applications in our testbed, it is in our future work to implement many other applications on the HydraOne platform and further study the energy savings of the coordinator.

The main difference between the baseline cases 1 and 2 is that for case 2 we only capture frame from one leopard camera. The RPLiDAR consumes 1.9W to connect via USB port and another 0.9W to scan, while the camera consumes 1.3W to connect via USB port and another 2.2W of capturing frames. Therefore, capturing frame from only one camera can save 2.2W of power dissipation. On the other hand, the power dissipation of case 2 is 1.3W lower than case 1, thus the power overhead of publishing frames from camera is 0.9W. However, after the camera frames are published, if an additional application requires the same camera frame, then the energy of adding another camera can be saved. Therefore, the sensor buffer can achieve energy saving in the multiple-application scenario. Overall, the E2M saves 4.5W (i.e., 18.5W-14W) of power dissipation. Considering the case 1 as the total power dissipation of the computing platform, then E2M saves up to 4.5W/18.5W=24.3% of the power dissipation of the computing platform. Comparing with the total power dissipation of HydraOne which is 39.1W, E2M saves 4.5W/39.1W=11.5% of the power dissipation. Extra 11.5 percent of battery time means the platform can run extra 14 minutes before being recharged.

Table 3: The descriptions of four cases.

Case #	Description
1	Baseline: HydraNet + MobileNet-SSD + Two cameras + RPLidar
2	Case 1 + sensor buffer
3	Case 1 + sensor buffer + waiting time
4	E2M (Case 1 + sensor buffer + waiting time + coordinator)

Figure 8(c) shows the power dissipation of the above four baseline cases when the GPU utilization ratio is set as 0.8. Comparing case 1 to 4 of Figure 8(b) and Figure 8(c), it can be found that the power dissipation decreases by 0.1W to 0.3W when the GPU utilization ratio increases from 0.2 to 0.8. The reason is that using GPU other than CPU for model inference accelerates the matrix computation and decreases the average power dissipation. For the power dissipation saving, E2M saves power dissipation by 4.4W and it is 24% of the power dissipation of the computing platform.

These experiments show that E2M can reduce the computational energy consumption by buffering the sensor data, using the applications' waiting time, and by coordinating the application execution.

6.3 Performance Analysis

In order to achieve high stability and availability for the AMRs, performance is an essential factor to consider. In this paper, the performance of the system is defined based on the applications running on top of HydraOne: HydraNet for free space detection

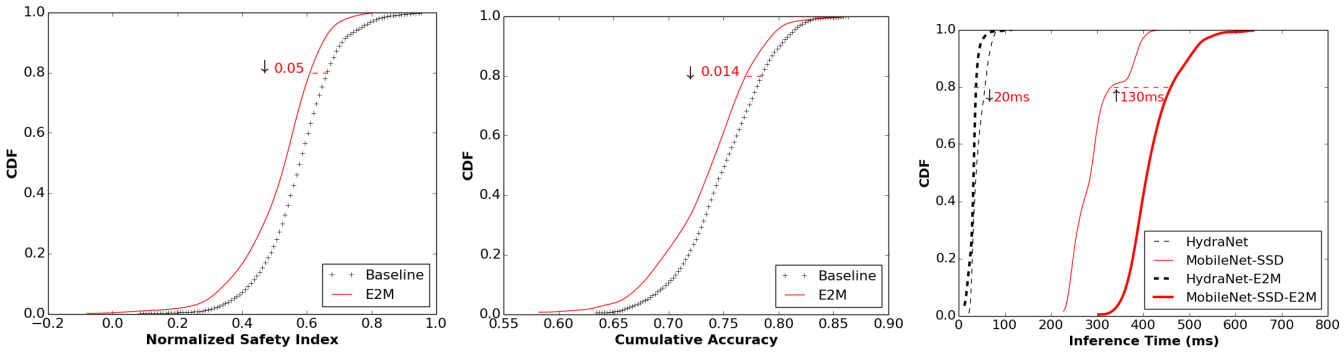


Figure 9: (a) The experiment results of normalized safety index. The higher the better. (b) The experiment results of normalized CA. The higher the better. (c) The experiment results of model inference time.

and MobileNet-SSD for object detection. The performance of E2M is discussed in three aspects: normalized safety index, normalized cumulative accuracy (CA), and model inference latency. Based on the implementation on HydraOne platform, we run HydraOne indoor with the same route for several hours to collect the safety index and CA. The CDF of the results of normalized safety index, normalized CA, and model inference latency are shown in Figure 9(a), 9(b), and 9(c), respectively.

Because the waiting time affects the frequency of model inference which causes the degradation of performance, we compare the performance of E2M with the baseline of no waiting time. In Figure 9(a), we compare the results of normalized safety index of E2M with no waiting time. Our purpose is to evaluate the average difference between the baseline's safety curve and E2M's safety curve. As one can observe in the figure 9(a) and 9(b), the E2M CDF curves are slightly shifted to the left compared to the baseline CDF curve. Thus, the difference between the baseline's safety/accuracy and the E2M safety/accuracy is about the same for all the percentiles. We can observe that the normalized safety index decreases by 0.05 compared with no waiting time case. This corresponds to just 7.9% of the normalized safety index with no waiting time. In Figure 9(b), we compare the results of normalized CA with no waiting time. We can observe that the normalized CA decreased by 0.014 compared with the no waiting time case, which corresponds to just 1.84% of the normalized CA with no waiting time.

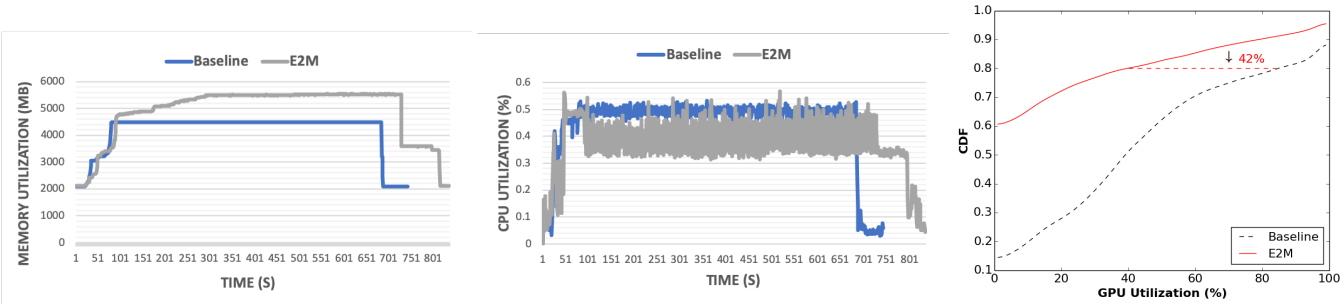
Another important performance metric is model inference latency. In Figure 9(c), we compare the model inference latency of HydraNet and MobileNet-SSD before and after implementing E2M. After applying E2M, the model inference latency of HydraNet decreases by 20ms while that of the MobileNet-SSD increases by 130ms in 80% of the data executions. The decrease of latency for HydraNet is owing to the ROS's publisher/subscriber mechanism, which decomposes the capturing frame and processing frame and allow them to run in parallel. Compared with capturing the frames from the camera after the model inference of last frame, the frames are get from the sensor buffer and processed in parallel. For the increase of latency for MobileNet-SSD, it's owing to the design of E2M. From the ROS framework of E2M in Figure 7, we can see the the ROS node */mobilenet* subscribes to the message of */image*, */scan*, and */control_hydra_time*, which means for each cycle of model inference it needs synchronize these three messages. The */control_hydra_time* message is generated by ROS node */control_time* based on */control_hydra* message from */hydranet*. However, after the ROS node */coordinator* publishes the coordinated waiting time to ROS node */camera*, the waiting time will be executed so the */image* will be delayed. After that, the ROS node */hydranet* needs to conduct model inference to get the */control_hydra* message, so the model inference of HydraNet is added to the delay. Therefore, the delay of model inference of MobileNet-SSD can be as high as 130ms. The effect of latency in inference can be resolved by the waiting time because the delay affects the performance and the performance affects the waiting time of each application.

These experiments show the E2M can achieve high energy savings with little performance degradation because it considers the desired performance within the decisions of the waiting times.

6.4 Resource Consumption

In addition to the energy overhead discussed in power dissipation section, the resource consumption of the system is also very important for the availability of the system. In this paper, we consider the system resource consumption by analyzing memory, CPU, and GPU usage. As the HydraOne's computing platform is a Jetson TX2 board, we use the tool called *tegrastats* to get the utilization of memory, CPU, and GPU. The GPU memory utilization ratio is set as 0.8.

The experiment results of memory, CPU, and GPU usage before and after applying E2M are shown in Figure 10(a), 10(b), and 10(c), respectively. From Figure 10(a), we can observe that the memory utilization increases by nearly 1GB after applying E2M. One reason for the increase of memory usage is that the ROS's publisher/subscriber mechanism contributes to many replicas of the same data, which is the main reason why E2M currently needs 1GB more of memory usage. On the other hand, the total system memory of Jetson TX2 board is 8GB, thus E2M uses 13% of the total system memory, which could be acceptable. Nevertheless, we plan to upgrade to ROS 2.0, which introduces shared memory to reduce the



(a) (b) (c)

Figure 10: The comparison of Baseline and E2M in: (a) the memory utilization, (b) the CPU utilization, (c) The GPU utilization.

system memory overhead of ROS topics. This should considerably lower the E2M memory overhead.

For the CPU utilization, as Jetson TX2 has six CPU cores, at each timestamp we calculate the summation of six cores' CPU utilization as the overall CPU utilization. From the results in Figure 10(b), we can see that the CPU utilization with E2M is lower than the baseline by 10% on average. The decrease of CPU usage is owing to the application of the waiting time. Specifically, the waiting time decreases the model inference frequency, which, in turns, leads to a decreased CPU utilization.

The CDF of GPU utilization of the baseline and E2M are shown in Figure 10(c). Considering the 80 percentile, we can observe that E2M can decrease the GPU utilization by 42%, which is owing to the introduce of waiting time. In addition, we can observe that the possibility for zero GPU utilization with E2M is over 60%. This phenomenon is caused by the coordination of waiting time across concurrent applications. When the waiting times of HydraNet and MobileNet-SSD are coordinated, the possibility for GPU to go to the sleep state is increased which contributes to more energy savings.

These experiments show that the execution of E2M does not causes new bottlenecks in the resource utilization of AMRs. On the contrary, in many cases such as CPU and GPU it contributes to reduce their activity.

7 DISCUSSION

Although E2M achieves nearly 24% of energy saving with less than 8% performance degradation, there are still some limitations owing to the design and targeted scenario. In this section, we discuss the limitations of E2M in three aspects: generality of E2M, multiple applications coordination, and random process of model inference.

7.1 Generality of E2M

E2M is a middleware designed mainly for autonomous mobile robots used for industry purposes. E2M is able to be deployed in any ROS-based AMRs. New applications can be implemented as plugins. For example, if we want to add the application of avoiding unexpected obstacles, we can change the way the performance metric for the navigation application is calculated. According to [22], we can introduce a distance space metric, which suggests that AMR applications need to respond within a certain response time given by: Response_space(m)/speed(m/s). For example, if we want a new decision every 1m at most and the current speed is 40km/h, then the desired response time is 90ms. We could use this as performance metric, without changing the described E2M algorithm.

For real autonomous vehicles, they have several commonalities with AMRs, including the need of analyzing camera images for object detection and driving. However, real environment is so complicated that even the most complex deep learning model may fail. Hence, the waiting time of the model inference may cause a considerable performance degradation. In addition to this, the multiple cameras on-board with multiple applications running in parallel, the coordination of sensor access and the coordination of the application executions can contribute to achieve energy savings even in the autonomous vehicle case.

7.2 Multiple Applications Coordination

In this paper, we only consider the coordination of concurrent applications in one cycle, which makes the case becomes easier because the coordination is just a local optimal solution. However, if we want to find the global optimal solution that achieves the most energy saving with guaranteed performance, we need to predict the performance changing, which can very challenging. In addition, as we have discussed in the performance analysis of the model inference latency, ROS node's messages will contribute to the delay of the message. Finally, the information that the coordinator gets can be all out-of-date. These problems may make it hard for E2M to save energy while guaranteeing performance. One solution is to use the newer ROS 2.0 rather than the currently used ROS, which introduces shared memory to improve the throughput and latency of data-sensitive communications.

7.3 Random Process of Model Inference

From Figure 8(a), we can observe that the inference time is a random process. Although the model is fixed and the input images are all from the same scenario, the inference time can vary from 21ms to 88ms for HydraNet and 225ms to 434ms for MobileNet-SSD. The random process of model inference time makes the waiting time of each application become a random process. This randomness makes it difficult for us to predict how long the model inference and the waiting time can be. Hence, the inference time and the waiting

time of each application also needs to be dynamic, which makes the coordination of concurrent applications challenging.

8 CONCLUSION

Autonomous mobile robots (AMRs) have been widely utilized in industry to execute various on-board computer-vision applications. Most of the applications involve the analysis of camera images through trained deep learning models. In this paper, we have first analyzed the breakdown of energy consumption for the execution of computer-vision applications on AMRs and discovered three main root causes of energy inefficiency: uncoordinated access to sensor data, performance-oriented model inference execution, and uncoordinated execution of concurrent jobs. In order to fix these three inefficiencies, we have proposed E2M, an energy-efficient middleware software stack for autonomous mobile robots. First, E2M regulates the access of different processes to sensor data, e.g., camera frames, so that the amount of data actually captured by concurrently executing jobs can be minimized. Second, based on a predefined per-process performance metric (e.g., safety, accuracy) and desired target, E2M manipulates the process execution period to find the best energy-performance trade off. Third, E2M coordinates the execution of the concurrent processes to maximize the total contiguous sleep time of the computing hardware for maximized energy savings. We have implemented a prototype of E2M on a real-world AMR. Our experimental results show that E2M leads to 24% energy savings for the computing platform, which translates into an extra 11.5% of battery time and 14 extra minutes of robot runtime, with a performance degradation lower than 7.9% for safety and 1.84% for accuracy.

ACKNOWLEDGEMENT

This work is supported in part by National Science Foundation (NSF) grant IIS-1724227.

REFERENCES

[1] 2019. Robot Operating System(ROS), Powering the World's Robots. (2019). https://www.ros.org/

[2] Christos Anagnostopoulos, Stathes Hadjiefthymiades, and Kostas Kolomvatsos. 2016. Accurate, Dynamic, and Distributed Localization of Phenomena for Mobile Sensor Networks. *ACM Trans. Sen. Netw.* 12, 2, Article 9 (April 2016), 59 pages. https://doi.org/10.1145/2882966

[3] and, Y. C. Hu, and C. S. G. Lee. 2004. Energy-efficient motion planning for mobile robots. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, Vol. 5. 4344–4349 Vol.5. https://doi.org/10.1109/ROBOT.2004.1302401

[4] D. Avola, G. L. Foresti, L. Cinque, C. Massaroni, G. Vitale, and L. Lombardi. 2016. A multipurpose autonomous robot for target recognition in unknown environments. In *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*. 766–771. https://doi.org/10.1109/INDIN.2016.7819262

[5] John A. Broderick, Dawn M. Tilbury, and Ella M. Atkins. 2014. Optimal coverage trajectories for a UGV with tradeoffs for energy and time. *Autonomous Robots* 36, 3 (01 Mar 2014), 257–271. https://doi.org/10.1007/s10514-013-9348-x

[6] F. de Lucca Siqueira, P. Della Mea Plentz, and E. R. De Pieri. 2016. A fuzzy approach to the autonomous recharging problem for mobile robots. In *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*. 1065–1070. https://doi.org/10.1109/FSKD.2016.7603326

[7] Luigi Di Puglia Pugliese, Francesca Guerriero, Dimitrios Zorbas, and Tahiry Razafindralambo. 2016. Modelling the mobile target covering problem using flying drones. *Optimization Letters* 10, 5 (01 Jun 2016), 1021–1052. https://doi.org/10.1007/s11590-015-0932-1

[8] S. Dogru and L. Marques. 2015. Energy Efficient Coverage Path Planning for Autonomous Mobile Robots on 3D Terrain. In *2015 IEEE International Conference on Autonomous Robot Systems and Competitions*. 118–123. https://doi.org/10.1109/ICARSC.2015.23

[9] F. Dressler and G. Fuchs. 2005. Energy-aware operation and task allocation of autonomous robots. In *Proceedings of the Fifth International Workshop on Robot Motion and Control, 2005. RoMoCo '05.* 163–168. https://doi.org/10.1109/ROMOCO.2005.201418

[10] H. Durmuş, E. O. Güneş, M. Kırcı, and B. B. Üstündağ. 2015. The design of general purpose autonomous agricultural mobile-robot: "AGROBOT". In *2015 Fourth International Conference on Agro-Geoinformatics (Agro-geoinformatics)*. 49–53. https://doi.org/10.1109/Agro-Geoinformatics.2015.7248088

[11] Murugappan Elango, Subramanian Nachiappan, and Manoj Kumar Tiwari. 2011. Balancing task allocation in multi-robot systems using K-means clustering and auction based mechanisms. *Expert Systems with Applications* 38, 6 (2011), 6486 – 6491. https://doi.org/10.1016/j.eswa.2010.11.097

[12] Josué Feliu, Julio Sahuquillo, Salvador Petit, and Jose Duato. 2016. Perf&Fair: A Progress-Aware Scheduler to Enhance Performance and Fairness in SMT Multicores. *IEEE Trans. Comput.* 66, 5 (2016), 905–911.

[13] Adrian Garcia-Garcia, Juan Carlos Saez, and Manuel Prieto-Matias. 2018. Contention-Aware Fair Scheduling for Asymmetric Single-ISA Multicore Systems. *IEEE Trans. Comput.* 67, 12 (2018), 1703–1719.

[14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[15] Christian Henkel, Alexander Bubeck, and Weiliang Xu. 2016. Energy Efficient Dynamic Window Approach for Local Path Planning in Mobile Service Robotics**This work was conducted at the University of Auckland, Auckland, New Zealand. *IFAC-PapersOnLine* 49, 15 (2016), 32 – 37. https://doi.org/10.1016/j.ifacol.2016.07.610 9th IFAC Symposium on Intelligent Autonomous Vehicles IAV 2016.

[16] Jason M Hirst, Jonathan R Miller, Brent A Kaplan, and Derek D Reed. 2013. Watts up? pro ac power meter for automated energy recording. (2013).

[17] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR* abs/1704.04861 (2017), 9. arXiv:1704.04861 http://arxiv.org/abs/1704.04861

[18] Ravindra Jejurikar and Rajesh Gupta. 2006. Optimized slowdown in real-time task systems. *IEEE Trans. Comput.* 55, 12 (2006), 1588–1598.

[19] A. Jevtic, Á. Gutierrez, D. Andina, and M. Jamshidi. 2012. Distributed Bees Algorithm for Task Allocation in Swarm of Robots. *IEEE Systems Journal* 6, 2 (June 2012), 296–304. https://doi.org/10.1109/JSYST.2011.2167820

[20] N. Kamra, T. K. S. Kumar, and N. Ayanian. 2018. Combinatorial Problems in Multirobot Battery Exchange Systems. *IEEE Transactions on Automation Science and Engineering* 15, 2 (April 2018), 852–862. https://doi.org/10.1109/TASE.2017.2767379

[21] B. Kannan, V. Marmol, J. Bourne, and M. B. Dias. 2013. The Autonomous Recharging Problem: Formulation and a market-based solution. In *2013 IEEE International Conference on Robotics and Automation*. 3503–3510. https://doi.org/10.1109/ICRA.2013.6631067

[22] Shinpei Kato, Eijiro Takeuchi, Yoshio Ishiguro, Yoshiki Ninomiya, Kazuya Takeda, and Tsuyoshi Hamada. 2015. An open approach to autonomous vehicles. *IEEE Micro* 35, 6 (2015), 60–68.

[23] D. Lee, S. A. Zaheer, and J. Kim. 2015. A Resource-Oriented, Decentralized Auction Algorithm for Multirobot Task Allocation. *IEEE Transactions on Automation Science and Engineering* 12, 4 (Oct 2015), 1469–1481. https://doi.org/10.1109/TASE.2014.2361334

[24] Dong-Hyun Lee. 2018. Resource-based task allocation for multi-robot systems. *Robotics and Autonomous Systems* 103 (2018), 151 – 161. https://doi.org/10.1016/j.robot.2018.02.016

[25] Jonathan Lenchner, Canturk Isci, Jeffrey O. Kephart, Christopher Mansley, Jonathan Connell, and Suzanne McIntosh. 2011. Towards Data Center Self-diagnosis Using a Mobile Robot. In *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC '11)*. ACM, New York, NY, USA, 81–90. https://doi.org/10.1145/1998582.1998597

[26] Leopard Imaging. 2019. Leopard Imaging AR023Z. (2019). https://leopardimaging.com/product/li-usb30-ar023zwdr/

[27] J. R. LeSage and R. G. Longoria. 2016. Mobile system shutdown prevention via energy storage-aware predictive control. In *2016 American Control Conference (ACC)*. 6815–6820. https://doi.org/10.1109/ACC.2016.7526745

[28] Jesse Levinson, Jake Askeland, Jan Becker, Jennifer Dolson, David Held, Soeren Kammel, J Zico Kolter, Dirk Langer, Oliver Pink, Vaughan Pratt, et al. 2011. Towards fully autonomous driving: Systems and algorithms. In *2011 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 163–168.

[29] Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md E Haque, Lingjia Tang, and Jason Mars. 2018. The architectural implications of autonomous driving: Constraints and acceleration. In *ACM SIGPLAN Notices*, Vol. 53. ACM, 751–766.

[30] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. 2016. SSD: Single Shot MultiBox Detector. In *Computer Vision – ECCV 2016*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). Springer International Publishing, Amsterdam, The Netherlands, 21–37.

[31] Maja J. Matarić, Gaurav S. Sukhatme, and Esben H. Østergaard. 2003. Multi-Robot Task Allocation in Uncertain Environments. *Autonomous Robots* 14, 2 (01 Mar 2003), 255–263. https://doi.org/10.1023/A:1022291921717

[32] F. Michaud and E. Robichaud. 2002. Sharing charging stations for long-term activity of autonomous robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vol. 3. 2746–2751 vol.3. https://doi.org/10.1109/IRDS.2002.1041685

[33] NVIDIA Corporation. 2019. NVIDIA Jetson Platform. (2019). https://www.nvidia.com/en-us/autonomous-machines/embedded-systems.

[34] K. J. O'Hara, R. Nathuji, H. Raj, K. Schwan, and T. Balch. 2006. AutoPower: toward energy-aware software systems for distributed mobile robots. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* 2757–2762. https://doi.org/10.1109/ROBOT.2006.1642118

[35] M. Rappaport and C. Bettstetter. 2017. Coordinated recharging of mobile robots during exploration. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* 6809–6816. https://doi.org/10.1109/IROS.2017.8206600

[36] T. F. Roos and M. R. Emami. 2018. A Framework For Autonomous Heterogeneous Robot Teams. In *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV).* 868–874. https://doi.org/10.1109/ICARCV.2018.8581303

[37] Francois Sempé, Angelica Muñoz, and Alexis Drogoul. 2002. Autonomous Robots Sharing a Charging Station with no Communication: a Case Study. In *Distributed Autonomous Robotic Systems 5*, Hajime Asama, Tamio Arai, Toshio Fukuda, and Tsutomu Hasegawa (Eds.). Springer Japan, Tokyo, 91–100.

[38] J.Tran S.Han, J.Pool and W.J.Dally. 2015. Learning both weights and connections for efficient neural networks. (2015).

[39] Slamtec. 2019. RPLIDAR A2 Laser Range Scanner. (2019). https://www.slamtec.com/en/Lidar/A2

[40] SMP Robotics. 2019. Application of Autonomous Mobile Robots. (2019). https://smprobotics.com/application_autonomus_mobile_robots/

[41] Munehiro Takimoto, Mayu Mizuno, Masato Kurio, and Yasushi Kambayashi. 2007. Saving Energy Consumption of Multi-robots Using Higher-Order Mobile Agents. In *Agent and Multi-Agent Systems: Technologies and Applications*, Ngoc Thanh Nguyen, Adam Grzech, Robert J. Howlett, and Lakhmi C. Jain (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 549–558.

[42] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. 2008. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics* 25, 8 (2008), 425–466.

[43] Lucia Vacariu, Blasko Peter Csaba, Ioan Alfred Letia, George Fodor, and Octavian Cret. 2004. A multiagent cooperative mobile robotics approach for search and rescue missions. *IFAC Proceedings Volumes* 37, 8 (2004), 962 – 967. https://doi.org/10.1016/S1474-6670(17)32105-5 IFAC/EURON Symposium on Intelligent Autonomous Vehicles, Lisbon, Portugal, 5-7 July 2004.

[44] Yifan Wang, Liangkai Liu, Xingzhou Zhang, and Weisong Shi. 2019. HydraOne: An Indoor Experimental Research and Education Platform for CAVs. In *Proceedings of the 2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19).* USENIX Association, RENTON, WA.

[45] Y. Wang, W. Peng, and Y. Tseng. 2010. Energy-Balanced Dispatch of Mobile Sensors in a Hybrid Wireless Sensor Network. *IEEE Transactions on Parallel and Distributed Systems* 21, 12 (Dec 2010), 1836–1850. https://doi.org/10.1109/TPDS.2010.56

[46] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. 2017. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 5687–5695.

[47] S. Yu and C. S. G. Lee. 2011. Lifetime maximization in mobile sensor networks with energy harvesting. In *2011 IEEE International Conference on Robotics and Automation.* 5911–5916. https://doi.org/10.1109/ICRA.2011.5979588

[48] X. Zhou and D. Kinny. 2013. Energy-Based Particle Swarm Optimization: Collective Energy Homeostasis in Social Autonomous Robots. In *2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, Vol. 2. 31–37. https://doi.org/10.1109/WI-IAT.2013.87