2020 IEEE International Conference on Robotics and Automation (ICRA)
31 May - 31 August, 2020. Paris, France

Balancing Actuation and Computing Energy in Motion Planning

Soumya Sudhakar, Sertac Karaman, Vivienne Sze

***Abstract*— We study a novel class of motion planning problems, inspired by emerging low-energy robotic vehicles, such as insect-size flyers, chip-size satellites, and high-endurance autonomous blimps, for which the energy consumed by computing hardware during planning a path can be as large as the energy consumed by actuation hardware during the execution of the same path. We propose a new algorithm, called *Compute Energy Included Motion Planning* (CEIMP). CEIMP operates similarly to any other anytime planning algorithm, except it stops when it estimates further computing will require more computing energy than potential savings in actuation energy. We show that CEIMP has the same asymptotic computational complexity as existing sampling-based motion planning algorithms, such as PRM*. We also show that CEIMP outperforms the average baseline of using maximum computing resources in realistic computational experiments involving 10 floor plans from MIT buildings. In one representative experiment, CEIMP outperforms the average baseline 90.6% of the time when energy to compute one more second is equal to the energy to move one more meter, and 99.7% of the time when energy to compute one more second is equal to or greater than the energy to move 3 more meters.**

I. INTRODUCTION

There has been an increasing interest in low-power robotic platforms. These platforms are power constrained usually due to a small form factor, *e.g.,* Robobee [1], [2], RoboFly [3], Viper Dash [4] or a long deployment duration (*e.g.,* Slocum Glider [5]). While these attributes constrain the power of the robot, they open up the possibility for a wide range of applications such as energy-efficient persistent environmental monitoring [6], distributed networks of spacecraft for space exploration [7], and non-invasive medicine delivery [8].

While there exists an active research field into developing low-powered actuated and controlled robotic platforms [1], [2], [3], [9], [10], a similar degree of attention has not been paid to algorithms specifically developed for deployment on low-power platforms. In this paper, we study a class of problems where the energy a robot spends for actuation during its motion is comparable to the energy spent computing the path. Table I shows the analogies between these two tasks. For the motion execution task, a robot must move along a path with length l_a at a speed v_a while drawing motor power P_a and expending actuation energy E_a; for the motion-plan computation task, its computer must compute n number of nodes requiring l_c operations at a processing speed of v_c while drawing computing power P_c and expending computing energy E_c. We can think of l_c as length of the computation which is analogous to length of the path l_a.

Authors are with the Massachusetts Institute of Technology, Cambridge, MA 02139, USA. Emails: {soumyas, sertac, sze}@mit.edu. This work was funded by the National Science Foundation, Cyber-Physical Systems (CPS) program, through grant no. 1837212.

TABLE I

Actuation	Computing
Path length [m], $l_a(n)$	Num. of operations [ops], $l_c(n)$
Vehicle velocity [m/s], v_a	Processing speed [ops/s], v_c
Actuation power [W], $P_a(v_a)$	Computing power [W], $P_c(v_c)$
Actuation time[s], $t_a(l_a(n), v_a)$	Computing time[s], $t_c(l_c(n), v_c)$

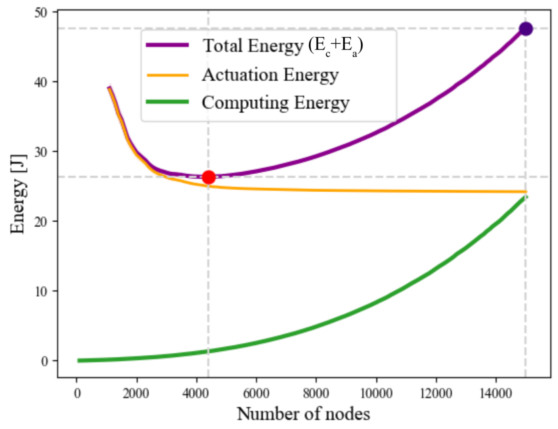


Fig. 1: Average total energy (computing + actuation) vs. nodes in PRM* tested on MIT Building 31 floor plan. The figure is compiled by averaging 500 trials.

In a conventional setting where $E_c \ll E_a$, only the first energy term for actuation energy in Eq. (1) is used to evaluate a motion planning algorithm since it dominates the total energy. Clearly, the path length $l_a(n)$ is decreasing with increasing nodes n computed. However, when the energy required for computing is not negligible compared to the energy required for actuation, we must consider the total energy of executing *and* computing a candidate solution:

$$\begin{aligned} E_t &= E_a + E_c \\ &= P_a(v_a)t_a(l_a(n), v_a) + P_c(v_c)t_c(l_c(n), v_c) \\ &= \frac{P_a(v_a)}{v_a}l_a(n) + \frac{P_c(v_c)}{v_c}l_c(n). \end{aligned} \tag{1}$$

Eq. (1) can be considered as the "cost-to-move" and the "cost-to-compute", making up the total energy E_t when considering actuation and computing energy. This trade-off is depicted in Fig. 1, assuming a $P_c = 7.5$W, and a robot platform that can go ≈ 0.9 m/s per 1 W spent. If we only look to minimize actuation energy, we end at the purple marker on the total energy curve, far from the minimum of the total energy. Minimizing actuation *and* computing energy involves stopping computation earlier at the red marker and accepting a larger "cost-to-move" for a lower "cost-to-compute".

There are several ways to have a dial to reduce E_c including stopping sampling at a certain number of nodes (decreases $l_c(n)$, $t_c(l_c(n), v_c)$) and switching to a simpler

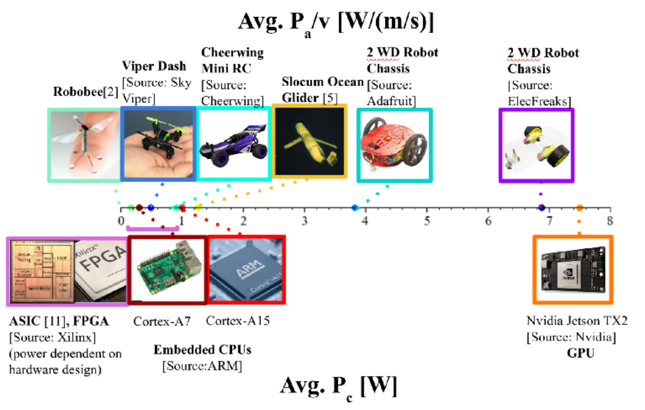

Avg. P_a/v [W/(m/s)]
Robobee[2]
Viper Dash [Source: Sky Viper]
Cheerwing Mini RC [Source: Cheerwing]
Slocum Ocean Glider [5]
2 WD Robot Chassis [Source: Adafruit]
2 WD Robot Chassis [Source: ElecFreaks]
0 1 2 3 4 5 6 7 8
ASIC [11], FPGA [Source: Xilinx] (power dependent on hardware design)
Cortex-A7
Cortex-A15
Embedded CPUs [Source:ARM]
Nvidia Jetson TX2 [Source: Nvidia] GPU
Avg. P_c [W]

Fig. 2: Actuation platforms and computing platforms on same scale of actuation power per m/s and computing power

algorithm (decreases $l_c(n)$, $t_c(l_c(n), v_c)$); if we allow $P_c(v_c)$ to increase, other options include increasing the clock frequency (decreases $t_c(l_c(n), v_c)$, increases v_c, $P_c(v_c)$) and parallelizing computation (decreases $t_c(l_c(n), v_c)$, $l_c(n)$ per core, increases v_c, $P_c(v_c)$) though the increase in $P_c(v_c)$ may negate the savings in $t_c(l_c(n), v_c)$ depending on the hardware architecture implementation. In this paper, we consider the first option, where, in order to balance $E_a + E_c$, we allow an algorithm to stop sampling at $n \leq n_{max}$ nodes instead of $n = n_{max}$.

The energy per meter is given by $P_a(v_a)/v_a$ and is a measure of the motor efficiency. The energy per operation is given by $P_c(v_c)/v_c$ and is a measure of the computing efficiency. For a fixed computer with a fixed clock frequency, we can look at computing power P_c and computing time $t_c(l_c(n), v_c)$ directly instead of computing efficiency $P_c(v_c)/v_c$ and number of operations $l_c(n)$. While $l_c(n)$ is simply a function of the algorithm complexity, computing efficiency is non-trivial to calculate since it is based on the hardware architecture design and the workload being run. Motor efficiency is analogous to computing efficiency, and is dependent on the dynamics and mass of the vehicle. For a fixed vehicle, we can assume a constant motor efficiency $P_a(v_a)/v_a$. Holding P_a, v_a, P_c, and v_c constant due to the fixed computer and fixed vehicle assumptions, we arrive at Eq. (2), where we have direct control over variable n and indirect control over $l_a(n)$, *i.e.,*

$$E_t = \frac{P_a}{v_a} l_a(n) + P_c t_c(l_c(n)). \tag{2}$$

Thus, we can choose to compute a larger n (higher $t_c(l_c(n))$ to reduce $l_a(n)$, hence lower "cost-to-move" at the expense of a higher "cost-to-compute". Alternatively, we can choose to compute a smaller n (lower $t_c(l_c(n))$ to reduce "cost-to-compute" at the expense of a higher $l_a(n)$, hence higher "cost-to-move".

The terms $P_c t_c(l_c(n))$ and $(P_a/v_a)l_a(n)$ indicate how to compare time spent computing and the distance to move. The computing power P_c is the constant that multiplies $t_c(l_c(n))$, and the motor efficiency P_a/v_a is the constant that multiplies $l_a(n)$. Thus, $E_c \ll E_a$ when $P_c \ll P_a/v_a$. When P_c is on a similar scale to P_a/v_a as in the case of low-power robotic platforms, it is important to consider E_c to minimize total energy.

Fig. 2 illustrates many examples of the P_a/v_a for robots and P_c for computers. Consider, for example, the Cheerwing mini RC car; with an actuation power of ≈ 4.9 W at ≈ 5.4 m/s [12], the car has a $P_a/v_a = 0.91$ W/(m/s). With an embedded CPU consuming on average 1 W, computing another 1 second is as expensive as moving another 1.1 meters. With an embedded GPU Nvidia Jetson TX2 consuming on average 7.5 W, computing another 1 second is as expensive as moving another 8.25 meters. In the 3D case, the Viper Dash can go at speeds up to ≈ 11.2 m/s at ≈ 5.5 W [4], making its $P_a/v_a = 0.49$ W/(m/s). With a computer averaging 1.0 W of power consumption, computing another 1 second is as expensive as moving 2.05 meters. When the Viper is equipped with a computer that consumes 7.5 W, computing for 1 second is as expensive as moving another 15.3 meters!

The main contribution of this paper is a framework to balance both energy spent on actuation and energy spent on computing in motion planning. Specifically, we introduce the *Compute Energy Included Motion Planning* (CEIMP) algorithm, which reasons about how long to compute to reduce total energy. CEIMP uses a Bayes estimator on the edges of a PRM* graph to predict which path in a different homotopic class may open up in the next batch of nodes and computes the expected path length $l_a(n)$ after an additional n nodes. CEIMP compares the expected savings in path length and the expected $t_c(l_c(n))$ after the next batch of nodes, and greedily decides to stop when "cost-to-compute" is greater than "cost-to-move". We implement and evaluate CEIMP on real 2D environments using floor plans of MIT buildings to show that CEIMP outperforms the baseline of minimizing actuation energy only.

II. RELATED WORK

Several works have looked at including contributions to total energy other than just path length such as wind, terrain, and communication costs [13], [14], [15], [16]. While this body of work adds the effect of different variables to total energy, it does not affect the "cost-to-compute" term of the planning computation task itself.

There has been a large body of work looking at reducing the "cost-to-compute" term towards efficient motion planning. In particular, variants of RRT* and PRM* [17] were proposed towards reducing $t_c(l_c(n))$, including the BIT* [18] and the FMT* [19]. Yet, to the best of our knowledge, this paper is the first work to examine the trade-off between E_a and E_c to reduce total energy for the motion planning problem.

The work presented in this paper is also related to exploring the obstacle space in motion planning. In this regard, Hauser [20] proposed a greedy algorithm to solve the minimum constraint removal problem and find the minimum obstacles to remove to find a feasible path. Hsu et al. [21] looked at dilating free space to find paths through narrow passageways to later repair. Bohlin et al. [22] proposed Lazy

PRM which searches for paths before doing collision checks to minimize computing on collision checks. These methods allow edges in a graph to be in obstacle space to improve performance in $l_a(n)$ [20], [21] or $t_c(l_c(n))$ [22], similar to the graph construction of CEIMP proposed here.

III. PROBLEM DEFINITION

Let X and X_{obs} denote the configuration space and the obstacle space. Let $X_{free} = X \setminus X_{obs}$. Given a start configuration $x_s \in X_{free}$ and a goal configuration $x_g \in X_{free}$, we aim to find a path $\sigma : [0,1] \to X_{free}$, starting in x_s, reaching x_g, avoiding X_{obs}, while minimizing energy:

$$\begin{aligned} \underset{\sigma}{\text{minimize}} \quad & E_a + E_c \\ \text{subject to} \quad & \sigma(0) = x_s, \sigma(1) = x_g, \sigma \in X_{free}. \end{aligned} \tag{3}$$

Note, the path σ is a function of n and the length of path σ is given by $l_a(n)$. This problem is challenging to solve to optimality. In particular, the "computational complexity" of the algorithm, *e.g.*, the number of operations executed to find a solution, is indeed embedded in its "performance metric," *i.e.*, the total energy function. In order to alleviate some of this complexity, we study a special case, in which an anytime planning algorithm is chosen, and the problem is to find the time to stop the algorithm towards minimizing total energy. Hence, the problem is to calculate n to solve the following:

$$\begin{aligned} \underset{n}{\text{minimize}} \quad & \frac{P_a}{v_a} l_a(n) + P_c t_c(l_c(n)) \\ \text{subject to} \quad & n \le n_{max}, \sigma(0) = x_s, \sigma(1) = x_g, \sigma \in X_{free}. \end{aligned} \tag{4}$$

This optimization problem can be applied to various grid-based or sample-based anytime motion planners. In this paper, we consider an anytime version of the PRM* as the underlying algorithm. A graph is defined as $G = (V, E)$, where V is a set of vertices and E is a set of edges. The PRM* is a sampling-based algorithm that finds asymptotically-optimal paths by sampling randomly in X_{free}, constructing a graph $G = (V, E)$ by connecting neighbor nodes in a ball whose radius decreases with increasing number of nodes, and running a search on those nodes to find the best path [17]. PRM* maintains a decreasing path length with respect to the number of nodes in the graph.

We define n^* as the number of nodes that satisfies the minimization in Eq. (4) and is the optimal number of nodes to compute before stopping. Our task is to estimate n^*. It is important to note that for a number of nodes n, the exact relationship between $l_a(n)$ and n is uncertain at any number of nodes less than n, such that the path length in n nodes before computing n nodes is uncertain. In other words, the actuation energy is uncertain until we spend the computing energy to compute it. Moreover, the relationship between $l_a(n)$ and n is difficult to model. Hence, spending computing resources to decide whether to devote more computing resources to planning adds a number of operations as overhead that do not contribute to lowering $l_a(n)$, which complicates the problem even further. Thus, with any algorithm estimating the optimal n^*, the total number of operations is equal to, $l_c(n) = \widetilde{l_c}(n) + \overline{l_c}(n)$, where $\widetilde{l_c}(n)$ is the number of operations used in the anytime motion planner to decrease $l_a(n)$ and $\overline{l_c}(n)$ is the number of operations used in CEIMP to estimate when to stop computing. To gain savings, it is necessary to predict n^* without introducing so much overhead as to negate any energy savings. In the next section, we discuss our proposed algorithm that greedily estimates n^* for a PRM*.

Algorithm 1: CEIMP

```
1  do
2      for $j = 1, 2, ..., b$ do
3          $x \leftarrow$ SampleFree; $G_{occl} \leftarrow$ AddToGraph($x$);
4      $l_a(n) \leftarrow$ SearchShortestPath($G_{occl}$);
5  while $l_a(n) \to \infty$
6  $continue \leftarrow true$;
7  while ($continue$) do
8      for $j = 1, 2, ..., b$ do
9          $x \leftarrow$SampleFree; $G_{occl} \leftarrow$AddToGraph($x$);
10     $G_{occl} \leftarrow$ UpdateProb; $E_c \leftarrow$ ComputeModel
11     $E_t \leftarrow$ SearchShortestPath($G_{occl}$);
12     $\widetilde{E_a} \leftarrow$ E-AStar($G_{occl}$); $\overline{E_a} \leftarrow$ Smoothing;
13     if $(\widetilde{E_a} + E_c) > E_t \wedge (\overline{E_a} + E_c) > E_t$ then
14         $continue \leftarrow false$;
```

IV. ALGORITHM

In this section, we propose *Computing Energy Included Motion Planning* (CEIMP), a greedy algorithm that predicts future computing cost and future actuation savings to estimate n^*. CEIMP is composed of three main parts as follows:

1) A graph constructor that samples in X_{free}. The constructor connects both feasible and infeasible edges $E \in X_{free} \cup X_{obs}$, and keeps track of feasible status.
2) A Bayesian filter-based estimator used to estimate the likelihood infeasible edges will be repaired. The results are used to search for the best expected path $\sigma \in X_{free} \cup X_{obs}$ that may be repaired with sampling.
3) A greedy decision-making rule that flags when to stop computing.

Graph construction is run for B total batches each of size b nodes, such that the i'th batch i involves adding b nodes to the graph at a time before searching for the current shortest path. Batch size b is a parameter that is chosen to balance performance in predicting n^* and overhead $\overline{l_c}(n)$ introduced. Estimation and decision making run in between batches of graph construction, such that CEIMP decides whether or not to continue computing after every batch. CEIMP is outlined in Algorithm 1, and we discuss each component of the algorithm next.

A. Graph Construction

CEIMP runs a modified PRM*, where infeasible edges are allowed to be added to the graph and marked as infeasible. Let an occluded graph $G_{occl} = (V, E, p_i(E))$ be defined by the set of all edges E, the set of all nodes V, and the set $p_i(E)$ that maps the probability at batch i that each edge $e \in E$ is feasible. For edge $e \in E$, if $p_i(e) = 1$, e is feasible; if

$p_i(e) < 1$, e is infeasible. For an infeasible edge, we initialize $p_i(e) = 0.5$ to reflect that while e is currently infeasible, it may be repaired in the future. After each batch i, $p_i(e)$ is updated for all infeasible edges, and only the most recent $p_i(e)$ is stored. A valid solution path is found by conducting a search only on feasible edges.

B. Estimation

The goal is to estimate the decrease in path length after computing an additional batch of nodes. In order to relate $l_a(n)$ and n, CEIMP predicts which currently infeasible paths are likely to open up with additional sampling. For each infeasible edge in G_{occl}, CEIMP estimates its state as a *real* edge or a *phantom* edge. We define a *real* edge as a currently infeasible edge that can be repaired by future sampling. For example, an infeasible edge that misses making a collision-free connection across a narrow passageway can likely be repaired with more samples. A *phantom* edge is a currently infeasible edge that cannot be repaired. For example, an infeasible edge that crosses a wall cannot be repaired with any number of nodes. We define repairing an infeasible edge $e = (v_1, v_2)$ with source node v_1 and destination node v_2 as sampling in a ball centered on the midpoint of e with diameter $\alpha||v_1 - v_2||_2$ and finding a path $\sigma \in X_{free}$ from v_1 to v_2. We can set α as a parameter that relates how much of an increase in path length we allow a repair to make.

We consider sampling around an edge as a measurement that can return two readings: repaired or unrepaired. The measurement model of the sampling is as follows. Let $S(e)$ be the state of the edge e that takes two values: real (s) or phantom ($\overline{s}$). Let $Y(e)$ be the measurement of the edge's state that takes on two values: repaired (r) or unrepaired ($\overline{r}$), and let a be the probability of repairing an edge given that it is a real edge. Then, the measurement model is given by,

$$\mathbb{P}(Y(e) = r) = \begin{cases} a & \text{if } S(e) = s \\ 0 & \text{if } S(e) = \overline{s} \end{cases} \tag{5a}$$

$$\mathbb{P}(Y(e) = \overline{r}) = \begin{cases} (1-a) & \text{if } S(e) = s \\ 1 & \text{if } S(e) = \overline{s}. \end{cases} \tag{5b}$$

This measurement model describes the process of sampling and repairing: if sampling finds a $\sigma \in X_{free}$, there is no uncertainty on whether that path exists. However, when sampling is unable to find a path $\sigma \in X_{free}$, there is still a chance $(1 - a)$ that with more sampling, a path $\sigma \in X_{free}$ could be found. Measurement model probability a is highly environment-dependent, and so when testing in real environments, a can be set to a constant parameter that can approximate the sampling process (*e.g.,* $a = 0.1$).

1) Bayesian Filter Update: In order to use past experience to estimate the state of an edge in G_{occl}, CEIMP uses a Bayesian filter. When attempting to repair an edge e succeeds, CEIMP updates $p_i(e) = 1$ along with the length of the repaired path. When attempting to repair an edge fails, CEIMP updates the probability through the following filter update equation, where $p_i(e) = \mathbb{P}(S(e) = s|Y_{1:i}(e) = \{\overline{r}, \overline{r}, ..., \overline{r}\})$, and $\mathbb{P}(Y(e) = \overline{r}|S(e) = s) = \mathbb{P}(\overline{r}|s)$. From our measurement model, we know $\mathbb{P}(\overline{r}|s) = (1 - a)$ and substitute to find,

$$\begin{aligned} p_i(e) &= \frac{\mathbb{P}(\overline{r}|s)p_{i-1}(e)}{\mathbb{P}(\overline{r}|s)p_{i-1}(e) + \mathbb{P}(\overline{r}|\overline{s})(1 - p_{i-1}(e))} \\ &= \frac{p_{i-1}(e) - ap_{i-1}(e)}{1 - ap_{i-1}(e)}. \end{aligned} \tag{6}$$

2) Expected-A:* Next, CEIMP runs a search to find the path with the lowest expected value on the updated G_{est}. We accomplish this through the implementation of Expected-A*. Expected-A* runs similarly to A*, with a new $f(\sigma)$ to order paths in the priority queue to reflect the expected cost-to-come and expected cost-to-go, *i.e.,*

$$\begin{aligned} f(\sigma_E) &= \mathbb{P}(\sigma_D)\,||\sigma_D||_2 \\ &+ [1 - \mathbb{P}(\sigma_D)]\mathbb{P}(\sigma_E)\,(||\sigma_E||_2 + ||g - u||_2) \\ &+ [1 - \mathbb{P}(\sigma_D)][1 - \mathbb{P}(\sigma_E)]l_a(n), \end{aligned} \tag{7}$$

where σ_D is the Euclidean distance path from start s to goal g and σ_E is the sub-path from start s to node u we are ordering in the queue. Note, $l_a(n)$ remains the length of the best feasible path. $\mathbb{P}(\sigma)$ is the joint probability that the edges that make up the path σ are all real and we repair each currently infeasible edge in the next batch of samples (each edge's state and each repair assumed independent) *i.e.,*

$$\mathbb{P}(\sigma) = \prod_{e \in \sigma, p_i(e) < 1} ap_i(e). \tag{8}$$

We show $f(\sigma)$ is admissible in Section V.

C. Decision Making

After estimation, CEIMP decides between choosing to stop computing or continue computing. To decide which option to select, CEIMP greedily looks at the expected total energy change of each option after the next batch of samples and chooses the option that has a larger decrease in total energy. The expected change in total energy is given by,

$$\begin{aligned} \mathbb{E}(\Delta E_t) &= \frac{P_a}{v_a}\left(\mathbb{E}(l_a(n+b) - l_a(n))\right. \\ &\left. + P_c\left(\mathbb{E}(t_c(l_c(n+b))) - t_c(l_c(n))\right)\right), \end{aligned} \tag{9}$$

where $\mathbb{E}[l_a(n + b)]$ is the expected value of the path length after computing b more samples and $\mathbb{E}[t_c(l_c(n + b))]$ is the expected time to compute b more samples.

The total energy change of stopping computing is known exactly: $\Delta E_t = 0$ J. We calculate the expected ΔE_t of continuing computing by using a computing energy model and the estimation of path length savings if we compute another batch of samples. The expected computing time $\mathbb{E}[t_c(l_c(n + b))]$ is set to the time to compute the previous batch $t_c(l_c(n)) - t_c(l_c(n - b))$, capturing that computing gets more expensive as more nodes are added to the graph.

CEIMP has two estimates of path length savings if we compute another batch of samples. The path length can decrease either due to finding a new path in a different homotopic class or by smoothing the current best path. Let $\mathbb{E}[\Delta\widetilde{E_t}]$ be the expected change in total energy from a homotopic class change and $\mathbb{E}[\Delta\overline{E_t}]$ be the expected change in total energy from smoothing. To calculate $\mathbb{E}[\Delta\widetilde{E_t}]$, CEIMP uses the result of the best expected path from Expected-A* to substitute for $\mathbb{E}(l_a(n+b))$ in Eq. 9. To calculate $\mathbb{E}[\Delta\overline{E_t}]$, CEIMP uses a smoothing model $k/n^{\frac{1}{d}}$ where d is the number of dimensions of the environment and k is an environment dependent constant. We can get an estimate of k using the improvement in feasible path length between batches such that, $k = \sqrt{n}(l_a(n-b) - l_a(n)) - 1/\sqrt{n+b}$. CEIMP then substitutes $\mathbb{E}(\overline{l_a}(n+b)) = k/\sqrt{n+b}$ to evaluate Eq. (9).

If both $\mathbb{E}[\Delta\widetilde{E_t}] \geq 0$ J and $\mathbb{E}[\Delta\overline{E_t}] \geq 0$ J, CEIMP decides to stop. Else, the expected total energy change of continuing is lower than the total energy change of stopping (0 J), and CEIMP will decide to continue. We can also run CEIMP with a biasing parameter $0 < \gamma < 1$. When CEIMP continues, γb number of nodes in the batch are biased to sample around either the best occluded path or the current best path based on whether CEIMP predicted larger savings from exploring a potential homotopic class change or from smoothing the current best path. CEIMP decides to

$$\begin{cases} \text{Stop} & (\mathbb{E}(\Delta\widetilde{E_t}) \geq 0) \wedge (\mathbb{E}(\Delta\overline{E_t}) \geq 0) \\ \text{Explore} & (\mathbb{E}(\Delta\widetilde{E_t}) < 0) \wedge (\mathbb{E}(\Delta\widetilde{E_t}) \leq \mathbb{E}(\Delta\overline{E_t})) \\ \text{Smooth} & (\mathbb{E}(\Delta\overline{E_t}) < 0) \wedge (\mathbb{E}(\Delta\overline{E_t}) < \mathbb{E}(\Delta\widetilde{E_t})). \end{cases} \tag{10}$$

V. ANALYSIS

We discuss the correctness of Expected-A* search and the computational complexity of CEIMP. We do not discuss optimality. The interlinked nature of n and $l_a(n)$ make it challenging, and we omit a rigorous discussion on optimality.

A. *Correctness of Expected-A**

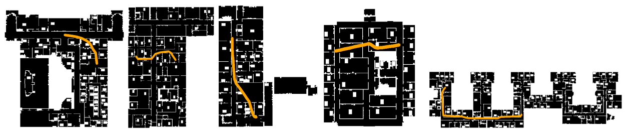
We show that Expected-A* heuristic $f(\sigma)$ is admissible and underestimates the true expected cost of the path when given a sub-path. We can rearrange terms in Eq. (7) to obtain,

$$f(\sigma) = c_1 + c_2 l_a(n) - c_2\mathbb{P}(\sigma_E)[l_a(n) - ||\sigma_E||_2 - ||g-u||_2], \tag{11}$$

where c_1, c_2 are constants such that $c_1 = \mathbb{P}(\sigma_D)||\sigma_D||_2$ and $c_2 = (1 - \mathbb{P}(\sigma_D))$. The first two terms of Eq. (11) are constants. The expression $c_2\mathbb{P}(\sigma_E)$ is monotonically decreasing as nodes are added to the sub-path since $\mathbb{P}(\sigma_E)$ is monotonically decreasing. The expression $(l_a(n) - ||\sigma_E||_2 - ||g-u||_2)$ is enforced to be positive by using branch-and-bound in search (never add paths to queue where $(||\sigma_E||_2 + ||g-u||_2 \geq l_a(n))$ and is monotonically decreasing as nodes are added to the sub-path since $(||\sigma_E||_2 + ||g-u||_2)$ is monotonically increasing. Therefore, the entire RHS of Eq. (11) is monotonically increasing as more nodes are added to the path, showing that $f(\sigma)$ is underestimating the expected cost of the full path and is admissible.

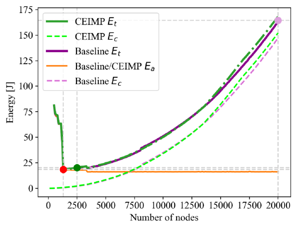
B. *Computational Complexity*

The PRM* runs in $O(n \log n)$ time [17]. The Bayes filter runs in $O(|E|)$ time. Expected-A* is the same constant factor complexity as A*, and is $O(n \log n)$. Checking if an edge has been repaired involves running A* on the nodes inside a ball sized by the edge and is $O(n_b \log n_b)$ where n_b is the number of nodes inside the ball. Overall computational complexity of CEIMP is therefore $O(n \log n)$.

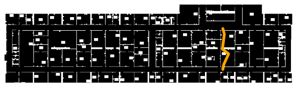


(a) 10 (b) 24-0 (c) 33 (d) 39 (e) W51

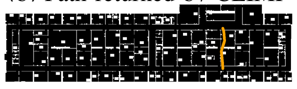
Fig. 3: Floor plans numbered by MIT campus building used in experiments with path to find



(a) Energy curves vs. n



(b) Path returned by CEIMP



(c) Path returned by baseline

Fig. 4: Single trial on Building 13 floor plan on Cortex-A15

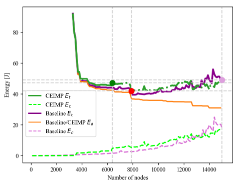
VI. IMPLEMENTATION

We use a floor plan dataset previously collected at MIT [23] to test on ten floor plans of campus buildings, five of which are shown in Fig. 3. There are additional rectangular obstacles added to the floor plans to add clutter.

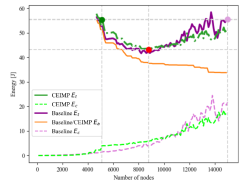
For experiments in Section VII testing CEIMP's performance, we test on the ARM Cortex-A7 and ARM Cortex-A15 embedded CPUs with clock frequencies of 1.2-1.6 GHz and 1.0-2.5GHz respectively and measured average power while running PRM* equal to 0.28 W and 2.33 W respectively. To obtain the results presented in Fig. 7, we also test on an Intel Xeon E5-2667 v4 with a clock frequency of 3.2GHz and which consumes up to 135W. To simulate the results swept on a range of computer efficiencies to motor efficiencies on one computer, we input to CEIMP a range of computing powers $P_c = 0.1$ to $P_c = 10.0$ W.

VII. COMPUTATIONAL EXPERIMENTS

We evaluate the performance of CEIMP in this section. For the results, we set P_a/v_a = 1W/(m/s). We run against two baselines: running an anytime PRM* for 20,000 nodes (baseline 1) and 15,000 nodes (baseline 2) and present both results. Note, performance of CEIMP is expected to show larger savings when the baseline is a larger number of nodes.



(a) Stopping near $\overline{n^*}$



(b) Greedy failure

Fig. 5: Single trial on Building 31 floor plan on Cortex-A7

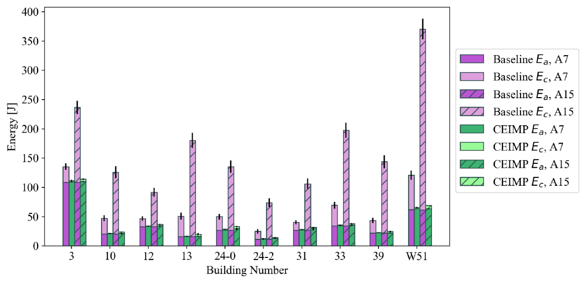


Fig. 6: Avg. energy over 1000 trials with $P_a/v = 1$ W/(m/s) on Cortex-A7 ($P_c = 0.28$W) and Cortex-A15 ($P_c = 2.33$W)

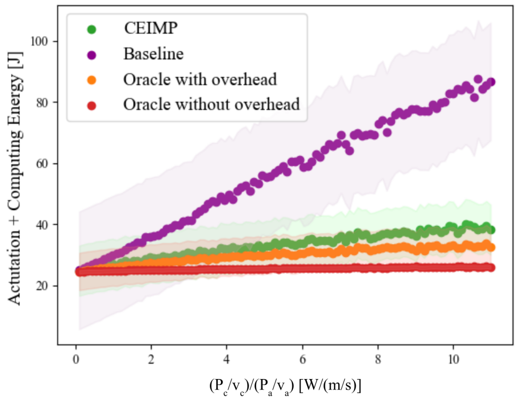


Fig. 7: Total average energy vs. ratio of P_c to motor efficiency with fixed v_c with standard deviation

Choosing too small of a baseline can risk poor performance in finding a feasible path in challenging environments. First, we look at examples of paths returned by CEIMP and the baselines in Figures 4 and 5 along with the total energy curves run on the ARM Cortex-A15 processor; we set $\gamma = 0$ and enforce that CEIMP samples the same nodes in the same order as the baseline to keep the actuation energy constant.

Note, PRM* incurs a cost at the end for searching the graph to find the best feasible path. Here, we run a search at every location on the curve and add that computing energy to the curve to show the total energy if computation was stopped at that point on the curve. The green marker on the CEIMP total energy curve is where CEIMP chooses to stop computing. To see how close CEIMP performed to finding the real n^*, we continue to sample randomly until maximum nodes are reached to obtain the projected CEIMP total energy if it had continued. The red marker in Fig. 4 and Fig. 5 shows $\overline{n^*}$ which is the optimal node to stop at given that CEIMP introduces overhead $\overline{l_c}(n)$; it is equal to the minimum of the green curve, and we call this point the *oracle-with-overhead*. $\overline{n^*}$ is different from n^*, which is the minimum of the baseline total energy curve; this theoretic limit would be minimum total energy given that we introduce no overhead $\overline{l_c}(n)$ to know to stop at n^*; we call this point the *oracle-without-overhead*.

We see in Fig. 4 an example of how CEIMP improves total energy, by estimating $\overline{n^*}$ closely and stopping before computing energy begins to increase E_t. As expected, in exchange for a lower E_c, the path length CEIMP returns is longer than the baseline, though overall E_t is reduced. CEIMP is a greedy algorithm, and thus is also prone to failure sometimes, as shown in Fig. 5b against baseline 2. The greedy failure rate decreases with increasing $P_c/(P_a/v_a)$. For example, on average for the Building 31 floor plan with a fixed v_c given by the Intel Xeon E5-2667 v4, looking at per-trial performance against the stricter baseline 2, CEIMP outperforms the average baseline 74.0% of the time at $P_c/(P_a/v_a) = 0.1$, 90.6% of the time at $P_c/(P_a/v_a) = 1.0$, and 99.7% of the time when at $P_c/(P_a/v_a) > 3.0$, with performance expected to be higher when v_c is lower.

Next, we look at how CEIMP performs across 10 different floor plans, averaged across 1000 trials for each environment on the Cortex-A7 and Cortex-A15. Fig. 6 shows that on average, CEIMP saves energy compared to baseline 1 on all floor plans on both the ARM Cortex-A7 and ARM Cortex-A15; this result also holds against baseline 2. On the ARM Cortex-A15, CEIMP saves 2.1x the total energy on Building 3 and 8.9x the total energy on Building 13 compared to baseline 1, translating to missions that can last 2.1x and 8.9x longer on the same battery. The variability in savings in different environments is likely due to the structure of the environment; if many homotopic classes exist, there is more to gain from continuing to compute which may make CEIMP's energy savings more modest, and vice-versa.

We end by looking at how well CEIMP performs on a range of different ratios of computing efficiency to motor efficiency $\frac{P_c}{v_c}/\frac{P_a}{v_a}$, which is comparing the energy needed to compute one more operation and the energy to move one meter for different computers and vehicle platforms. Fig. 7 shows the total energy of CEIMP and baseline 2, averaged across 100 trials. In addition, we compute the average *oracle-with-overhead* and *oracle-without-overhead* curves. We see that CEIMP on average consistently has a lower E_t than the baseline, and the savings increase as $\frac{P_c}{v_c}/\frac{P_a}{v_a}$ increases. Moreover, CEIMP behaves very similarly to the *oracle-with-overhead*, showing that while greedy, it is able to estimate $\overline{n^*}$ well in practice. CEIMP is bounded below by the *oracle-with-overhead* average as expected. Overall, these results show CEIMP's advantage in low-power motion planning.

VIII. CONCLUSION

In this paper, we identify a class of problems in motion planning where the *cost-to-compute* is comparable to the *cost-to-move*. For this class of problems, we reframe motion planning from minimizing only E_a to minimizing both $E_a + E_c$. We solve the problem posed by proposing the greedy algorithm CEIMP that predicts future actuation savings and future computation spent to find those actuation savings to decide if and when to stop computing. CEIMP shows improved performance over the baseline and looks to be promising solution to the low-power motion planning problem. Future work involves testing CEIMP onboard low-power actuated robotic platforms as well as identifying theoretical bounds on performance. For more details on experimental results, we recommend visiting the project website (https://lean.mit.edu/projects/CEIMP).

REFERENCES

[1] R. Wood, R. Nagpal, and G.-Y. Wei, “Flight of the robobees,” *Scientific American*, vol. 308, no. 3, pp. 60–65, 2013.

[2] N. T. Jafferis, E. F. Helbling, M. Karpelson, and R. J. Wood, “Untethered flight of an insect-sized flapping-wing microscale aerial vehicle,” *Nature*, vol. 570, no. 7762, p. 491, 2019.

[3] J. James, V. Iyer, Y. Chukewad, S. Gollakota, and S. B. Fuller, “Liftoff of a 190 mg laser-powered aerial vehicle: The lightest wireless robot to fly,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–8.

[4] “Sky viper dash.” [Online]. Available: http://sky-viper.com/dash/

[5] D. L. Rudnick, R. E. Davis, C. C. Eriksen, D. M. Fratantoni, and M. J. Perry, “Underwater gliders for ocean research,” *Marine Technology Society Journal*, vol. 38, no. 2, pp. 73–84, 2004.

[6] M. Dunbabin and L. Marques, “Robots for environmental monitoring: Significant advancements and applications,” *IEEE Robotics & Automation Magazine*, vol. 19, no. 1, pp. 24–39, 2012.

[7] K. Schilling, “Perspectives for miniaturized, distributed, networked cooperating systems for space exploration,” *Robotics and Autonomous Systems*, vol. 90, pp. 118–124, 2017.

[8] M. Sitti, “Miniature soft robots—road to the clinic,” *Nature Reviews Materials*, vol. 3, no. 6, p. 74, 2018.

[9] J.-S. Koh, E. Yang, G.-P. Jung, S.-P. Jung, J. H. Son, S.-I. Lee, P. G. Jablonski, R. J. Wood, H.-Y. Kim, and K.-J. Cho, “Jumping on water: Surface tension–dominated jumping of water striders and robotic insects,” *Science*, vol. 349, no. 6247, pp. 517–521, 2015.

[10] D. Rus and M. T. Tolley, “Design, fabrication and control of origami robots,” *Nature Reviews Materials*, vol. 3, no. 6, p. 101, 2018.

[11] A. Suleiman, Z. Zhang, L. Carlone, S. Karaman, and V. Sze, “Navion: a fully integrated energy-efficient visual-inertial odometry accelerator for autonomous navigation of nano drones,” in *2018 IEEE Symposium on VLSI Circuits*. IEEE, 2018, pp. 133–134.

[12] “Cheerwing 1:32 mini rc racing car.” [Online]. Available: www.cheerwing.com/rc-cars

[13] J. Ware and N. Roy, “An analysis of wind field estimation and exploitation for quadrotor flight in the urban canopy layer,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1507–1514.

[14] N. Ganganath, C.-T. Cheng, and K. T. Chi, “A constraint-aware heuristic path planner for finding energy-efficient paths on uneven terrains,” *IEEE Transactions on Industrial Informatics*, vol. 11, no. 3, pp. 601–611, 2015.

[15] Y. Yan and Y. Mostofi, “To go or not to go: On energy-aware and communication-aware robotic operation,” *IEEE Transactions on Control of Network Systems*, vol. 1, no. 3, pp. 218–231, 2014.

[16] Z. Liu, J. Dai, B. Wu, and H. Lin, “Communication-aware motion planning for multi-agent systems from signal temporal logic specifications,” in *2017 American Control Conference (ACC)*. IEEE, 2017, pp. 2516–2521.

[17] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[18] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, “Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 3067–3074.

[19] L. Janson, E. Schmerling, A. Clark, and M. Pavone, “Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions,” *The International Journal of Robotics Research*, vol. 34, no. 7, pp. 883–921, 2015.

[20] K. Hauser, “The minimum constraint removal problem with three robotics applications,” *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 5–17, 2014.

[21] D. Hsu, L. E. Kavraki, J.-C. Latombe, R. Motwani, S. Sorkin, *et al.*, “On finding narrow passages with probabilistic roadmap planners,” in *Robotics: the Algorithmic Perspective: 1998 Workshop on the Algorithmic Foundations of Robotics*, 1998, pp. 141–154.

[22] R. Bohlin and L. E. Kavraki, “Path planning using lazy PRM,” in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 1. IEEE, 2000, pp. 521–528.

[23] E. J. Whiting, “Geometric, topological & semantic analysis of multi-building floor plan data,” Ph.D. dissertation, Massachusetts Institute of Technology, 2006.