

The Swarm as a Service: Virtualization of Motion

Daniel Graff, Reinhardt Karnapke
 Communication and Operating Systems Group
 Technische Universität Berlin
 10587 Berlin, Germany
 Email: {daniel.graff, karnapke}@tu-berlin.de

Abstract—A defining aspect of Cyber-physical systems are the sensors and actuators of a computing system, which interact with the physical world. While sensors and actuators can take on many different forms, one type of actuator, namely motors, usually has the highest influence on the lifetime due to the high energy consumption. Therefore, it is necessary to reduce physical movement of robots as much as possible without reducing the quality of the application. In this paper we introduce virtual movement of applications, meaning that parts of an application can be executed on arbitrary robots that can reach the locations defined by the application programmer with minimal effort. As we assume that robots will be shared by multiple applications, programming virtual movement within the application is not only cumbersome, but impossible, as the location and status of a robot may be changed by a different application. Therefore, we provide a middleware abstraction which takes care of virtual movement and hides it from the application programmer.

I. INTRODUCTION

Cyber-physical systems (CPS) are systems in which the interaction between computing and non-computing elements, i.e., the physical world, is a basic property. Typically, the execution of applications for CPS is constrained in physical space. In general, applications can be categorized into three classes concerning physical space: *location-independent* (they do not require physical space for execution), *fixed location* (the execution is bound to a fixed location) and *alternating locations* (the execution of an application requires locomotion).

However, a “moving” application enables the following scenarios: depending on time and weather conditions, driving over a mountain pass is sometimes hazardous, i.e., rocks or snow might block certain road sections making the crossing impossible and require clearance first. Usually such incidents are reported by passengers which approach the hazardous areas. In order to achieve higher safety and be aware of incidents earlier in order to close the road and avoid passengers getting stuck, an application can be programmed that acts as a virtual vehicle that repeatedly moves over the mountain pass while taking pictures and measuring the temperature at certain time intervals. Such data can be evaluated in order to detect incidents. The same can be done for people hiking along a trail which goes through different areas and includes different elevations (climbing). Using such applications people know in advance about the difficulties of the trail (steep passages) and know weather conditions, air pressure, etc. in order to equip themselves appropriately.

In general, the application follows a virtual trajectory which consists of geographical points. The programmer specifies

which actions should be done at the respective points, i.e., measure temperature, take picture, etc. The other application example includes exploration of unknown territory, monitoring, sensing of attributes at certain points of interest.

The straightforward solution for application execution is to have full control of the required physical resources: allocate the resources, move the resource and execute instructions on the resource. There are several reasons why this is not recommended: Full control requires to own the resources and to operate the infrastructure. Adding more features to an applications requires more capabilities of the infrastructure, i.e., specialized hardware featuring various sensors and actuators. This is cost-intensive and also requires maintenance. Second, full control also introduces high complexity for the application programmer. All sensors and actuators as well as locomotion must be explicitly programmed by the application programmer in addition to concurrency and distribution.

Our solution is an abstraction layer for CPS, i.e., an operating system that controls and manages all resources while applications run on top of the system [1], [2]. The benefits are: sharing one infrastructure (costs are shared among the participants, this also includes maintenance), less programming effort necessary (suitable programming abstraction), energy efficiency, and providing more (virtual) resources than (physically) exist. Virtualization is a well known and efficient approach in many ways, e.g., virtual machines, virtual memory (paging) or processor virtualization (tasks are mapped to CPUs in a time sharing manner).

Figure 1 shows an example in which *one* application is programmed that performs monitoring tasks, e.g. temperature measurements, at different physical locations. The application is composed of several actions, each of which is bound to a certain location. The territory consists of multiple areas: water, rough and “smooth” (simple) terrain. Several robots are available in that area: UAVs (unmanned aerial vehicles), UGVs (unmanned ground vehicles) and USVs (unmanned surface vehicles). Every vehicle has different movement capabilities: USVs can only operate on water, UGVs can only operate on simple terrain and UAVs have arbitrary movement. Hence, vehicles have their preferred domain of operation.

As depicted in Figure 1(a), the application virtually moves along the “arrows”, i.e., first to the south and afterwards to the east. We define that the application is executed by a virtual entity. In particular, there are nine actions partitioned into three groups. The actions of each group shall be executed

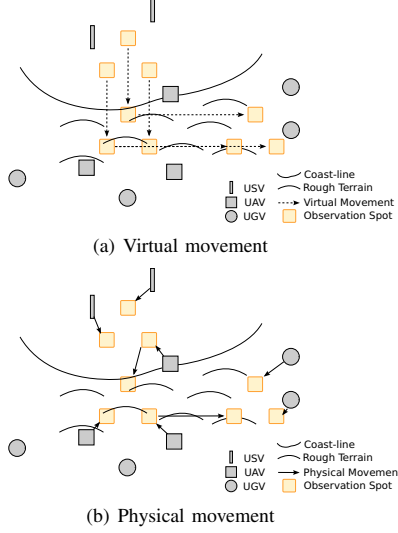


Fig. 1. Application movement

simultaneously. Therefore, the application is executed from three virtual entities that collectively move along the indicated path.

The virtual path of the application goes through different terrain types. Figure 1(b) shows the actual physical movement that is performed in order to execute the application which involves several vehicles. Executing multiple applications, the physical vehicles are used in a time sharing manner.

Advantages of this approach are:

- Introducing virtual movement reduces the real movement of the robots, enabling the execution of more applications in average and, thus, increasing the energy efficiency and/or lifetime of the system.
- No coordination necessary: The system computes the mapping of applications, in particular actions, to executing devices which is transparent for the programmer.
- No explicit device migration necessary: If an application is migrated from one device to another this is triggered by the system, and is transparent for the programmer.
- No control of locomotion necessary: If a device has to be physically moved, the system induces the movement by steering the devices actuators.

In this paper, we show how we use virtualization of movement in a CPS in order to reduce actual movement and, therefore, energy consumption. This approach enables the execution of more applications over a fixed amount of time or a longer lifetime of the system. The remainder of this paper is structured as follows: Section II describes our model of the world, which is used by the CPS. Section III explains the scheduling algorithm that maps actions to robots, i.e., the virtual movement is mapped to a sequence of physical movement parts. Section IV evaluates the algorithm, primarily by examining the amount of virtual and physical movement. Section V lists related work in this field and Section VI concludes the paper.

II. ASSUMPTIONS AND MODEL

This section states basic assumptions and presents the model of the world and the scheduler.

A. The Model of the World

The world has 3 dimensions: 2 space dimension (surface) and a time dimension. All entities in the world have their own geometry represented as a 2D polygon. Entities can be either static or dynamic: the former ones stay at a fixed location while the latter ones move over time. A function $f : T \rightarrow \mathbb{R}^2$ describes the spatio-temporal relation.

R denotes the set of mobile robots (dynamic entities) and O^s denotes the set of static obstacles (static entities). From the “perspective” of a robot r_i , all other robots $R \setminus \{r_i\}$ are considered as dynamic obstacles O^m .

B. Actions and ActionSuites

An action $a \in A$ is defined as a tuple (g, t_{min}, t_{max}, d) . The first element states the geometry g that consists of 2D points that mark the space window on a plain surface. The time-window is denoted by t_{min} and t_{max} . Together (g, t_{min}, t_{max}) describe the space-time-window in which a shall be executed while d denotes the execution time.

If an action a_i depends on another action a_j , then this is denoted as: $a_i \rightarrow a_j$; if $a_j \parallel a_i$, then a_j, a_i are independent, with $i \neq j$.

An ActionSuite $as \in AS$ is defined as a container for actions. An action a is assigned to exactly one suite: if $a \in as_i$, then $\forall as_j \in AS : a \notin as_j$, with $i \neq j$. Each action has to belong to a suite: $\forall a \in A : a \in AS$. All depending actions have to reside in the same suite. However, it is also possible to put non-depending actions in the same suite.

The elements of a suite as together with their dependencies form a directed, acyclic graph. The graph can be either connected or disconnected. A topological sorting on the graph is performed which generates the order in which the actions are scheduled.

C. Virtualization

In this section, we introduce the concept of virtualized resources: a virtual robot uses virtual movement in order to move. A virtual robot is mapped to a physical robot in a time sharing manner which performs physical movement in order to change its place.

Programmers define actions which are executed on virtualized sensors and actuators mounted on virtual robots, i.e., they program a specification. The system is responsible for finding a mapping of actions on virtualized sensors and actuators to physical ones (including locomotion). The amount of virtual robots is defined by the programmer.

- **Virtual Movement:** Virtual movement defines the motion of a virtual entity, e.g., a program. Virtual movement is existent if the program executing device physically moves itself or if the program is migrated to another device.
- **Physical Movement:** Physical movement defines the motion of a physical device.

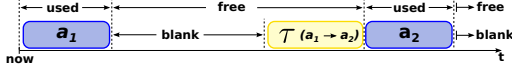


Fig. 2. Local schedule S_i of a robot.

D. Schedule

The objective of the scheduler is to compute a schedule that maps the actions $a_1, a_2, \dots, a_n$ to the robots under consideration of the spatio-temporal constraints of $a_1, a_2, \dots, a_n$. Actions are scheduled sequentially and in the order of the topological sorting. The topological sorting depends on the set of root actions A_{as}^r , the inter-action dependencies, and the spatio-temporal constraints.

For each action a , the scheduler computes a job (p, t, r) : $p \in g$ is the execution location, $t \in [t_{min}, t_{max} - d]$ is the start time and $r \in R$ is the executing robot. If r is not already at position p , then the scheduler computes a collision-free, spatio-temporal trajectory in order to move r to p . The trajectory consists of linear segments and is given by a list of spatio-temporal points $[(x_1, y_1, t_1), (x_2, y_2, t_2), \dots, (x_m, y_m, t_m)]$, where a segment is defined by a pair of points $[(x_i, y_i, t_i), (x_{i+1}, y_{i+1}, t_{i+1})]$. Let $\vec{x} = (x, y)$, then $\Delta\vec{x}_i = \vec{x}_{i+1} - \vec{x}_i$ and $\Delta t_i = t_{i+1} - t_i$, $t_{i+1} > t_i$ holds. The robot moves along the trajectory with velocity $v = \frac{|\Delta\vec{x}|}{\Delta t}$; the velocity is constant for a segment, but different segments may have different velocities: $[v_1, v_2, v_{m-1}]$.

Let $S^g = \{S_1, S_2, \dots, S_k\}$ be the global schedule, then S_i is the local schedule for robot r_i ; $k = |R|$. The initial schedule is the empty schedule: $S = \{\}$.

A schedule S consists of jobs: $S_i = \{j_1, j_2, \dots, j_l\}$. A job can be either a spatio-temporal action (a) or a spatio-temporal trajectory (τ). Figure 2 shows a simple schedule that consists of two action jobs and one movement job. Gaps between jobs are called *blank slots*. All spatio-temporal actions are (once scheduled) committed, i.e., the executing robot and the point in time is confirmed. In order to schedule new actions, all spatio-temporal trajectories can be arbitrarily replaced as long as they do not violate the execution of committed actions. Therefore, all gaps between actions are called *free slots*.

E. System Metrics

The system utilization $u(t_1, t_2)$, with $t_2 > t_1$, reflects the utilization in the interval $[t_1, t_2]$ and is defined as

$$u(t_1, t_2) := \frac{u_1(t_1, t_2) + u_2(t_1, t_2) + \dots + u_n(t_1, t_2)}{n} \quad (1)$$

with u_r being the local utilization of robot r . The utilization $u_r(t_1, t_2)$ is based on the job utilization $u_r^j(t_1, t_2)$ and the motion utilization $u_r^m(t_1, t_2)$:

$$u_r(t_1, t_2) := u_r^j(t_1, t_2) + u_r^m(t_1, t_2) \quad (2)$$

The job utilization u_r^j states the fraction of u which is required in order to execute actions and is defined as

$$u_r^j(t_1, t_2) := \frac{\sum_{a \in A_r \mid a \in [t_1, t_2]} dur(a)}{\Delta_t} \quad (3)$$

A_r is the set of all jobs associated with robot r , $dur(a)$ states the duration of job a and $\Delta_t := t_2 - t_1$. The motion utilization u_r^m defines the fraction of u which is required for physical movement of the robots and is given by

$$u_r^m(t_1, t_2) := \frac{\sum_{\tau \in T_r \mid v(\tau) > 0} dur(\tau)}{\Delta_t} \quad (4)$$

T_r is the set of all trajectories τ associated with robot r . A trajectory may consist of multiple segments, each of which may also have different velocities. The function $dur(\tau)$ delivers the time duration of the trajectory length for all segments which have a velocity greater than 0 ($v(\tau) > 0$).

The virtual movement s_{virt} is given by

$$s_{virt} = \sum_{vr=1}^{|VR|} \left(\|\vec{x}^{vr} - \vec{x}_{a_1}^{vr}\| + \sum_{i=1}^{k-1} \|\vec{x}_{a_i}^{vr} - \vec{x}_{a_{i+1}}^{vr}\| \right) \quad (5)$$

and indicates the total length that $|VR|$ virtual robots have to move in order to execute a set of k actions where $\vec{x}_{a_i}^{vr}$ defines the execution location of the i -th action of the virtual robot vr ; k depends on the actual amount of actions that vr has been assigned to. The virtual trajectory (of one virtual robot vr) starts at the location $\vec{x}^{vr}$ which is the location of a suitable (physical) robot r : $\vec{x}_i^r$ and depends on the ordering (sequence) of actions. The physical movement s_{phys} is given by

$$s_{phys} = \sum_{r=1}^{|R|} \sum_{i=0}^{n-1} \|\vec{x}_i^r - \vec{x}_{i+1}^r\| \quad (6)$$

and indicates the actual physical movement necessary in order to execute the actions. R denotes the set of robots. Each robot has assigned trajectories which consist of segments. Let n be the total number of segments per robot, then s_{phys} is the total (physical) length of all segments, i.e., the path length that the robot has to move. The efficiency is given by

$$ef = 1 - \frac{s_{phys}}{s_{virt}} \quad (7)$$

If $ef \in (0, 1)$, then physical movement could be replaced by virtual movement which is a benefit for the system. Higher values represent higher benefits.

III. SCHEDULING

The scheduler consists of two parts: the *job scheduler* and the *trajectory planner*. The job scheduler obtains as input the tuple of an action $a = (g, t_{min}, t_{max}, d)$. Using a heuristic (static and dynamic obstacles are neglected in this step), it determines a *free slot* ρ of a robot in order to reach the execution location $\vec{x}_a$ of a . Afterwards, the parameters $\vec{x}_a$, ρ , a are provided to the trajectory planner which computes the actual trajectory (in this step collision avoidance with static and dynamic obstacles is considered).

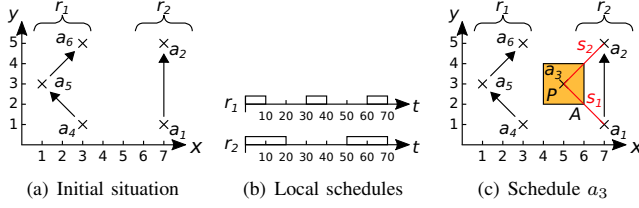


Fig. 3. Schedule new action

A. Job Scheduling

First, the job scheduler samples a location candidate $p^* \in g \setminus \{O^s\}$ which is an interior point of the area spanned by the space constraint g . The candidate must not lie in an obstacle O^s . If no schedule exists using p^* then the area spanned by g is divided into four sub-areas $g_1, \dots, g_4$ and the procedure is repeated recursively. A maximum recursion depth prevents the algorithm from going into an endless loop. A suitable location p^* marks the execution location of a : $\vec{x}_a = p^*$. The action a needs to be assigned to a robot r . In particular, the scheduler determines a free slot ρ of a robot, which is given by the time interval $[t_1, t_2]$, that enables to plan the action in that interval. At each beginning of a free slot, the robot might be at different locations (since it already executed an action) and, thus, selecting a certain slot influences the overall trajectory length that the robot has to follow. However, ρ must be chosen such that the robot is able to move to the location $\vec{x}_a$, execute a and, finally, move to the location $\vec{x}_{a_{i+1}}$ where the next scheduled action a_{i+1} is executed. In order to execute a , the length of the robots trajectory is increased. The fraction of the trajectory which states the additional movement is defined as a detour for r . In order to have minimum movement, the scheduler selects the slot ρ which minimizes the detour.

Figure 3(a) visualizes a 2D surface with two robots r_1 and r_2 ($v_{max} = 0.5$). Both of them have already scheduled jobs: r_1 has three jobs and r_2 has two jobs. The arrows indicate the direction of movement to the execution location of the next job. Figure 3(b) shows the local schedules $\mathcal{S}_1 = \{a_4, a_5, a_6\}$ and $\mathcal{S}_2 = \{a_1, a_2\}$ which state only actions jobs (j^a). For simplification, the trajectories are omitted since they will be replaced when scheduling a new action. Next, a new action a_3 shall be scheduled (Figure 3(c)). The spatial constraints are $x = [4, 6]$ and $y = [2, 4]$. The temporal constraint is given by $t = [10, 60]$. The location candidate was set to $p^* = (5, 3)$. The only free slot that fits is the one from r_2 : $\rho = [20, 50]$. According to the schedule $\mathcal{S}_2$ the robots trajectory is given by $\vec{x}_{a_1} \rightarrow \vec{x}_{a_2}$ and has a length of $s = \|\vec{x}_{a_1} - \vec{x}_{a_2}\| = 4$. Scheduling a_3 and assigning it to r_2 results in a modified trajectory: $\vec{x}_{a_1} \rightarrow \vec{x}_{a_3} \rightarrow \vec{x}_{a_2}$. This results in the following distances: $s_1 = \|\vec{x}_{a_1} - \vec{x}_{a_3}\| = \sqrt{8}$ and $s_2 = \|\vec{x}_{a_3} - \vec{x}_{a_2}\| = \sqrt{8}$. With $v_{max} = 0.5$, the robot needs approximately 11.32 time units for the movement. The execution time of a_3 is given by $d = 10$. The total required time is 21.32 time units. Therefore, the only possible slot is the one of r_2 . The length of the detour is: $\|s - (s_1 + s_2)\| = 1.66$.

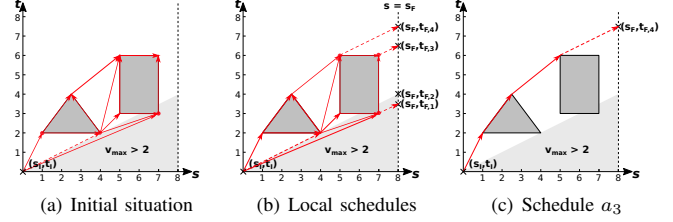


Fig. 4. Forbidden regions and velocity profiles

B. Trajectory Planning

In this step, the collision-free spatio-temporal trajectory which avoids static as well as dynamic obstacles is computed. For this, the parameters from the job scheduler ($\vec{x}_a, \rho, a$) are used. This section, hence, addresses the trajectory planning problem (TPP). We use the approach of the path-velocity-decomposition presented in [3] which splits the problem into the static path planning problem (PPP) and the dynamic velocity planning problem (VPP). The main reason for performing the decomposition is to face the complexity of the problem. In general, “the complexity of the PPP is exponential in the number of degrees of freedom, which for a point robot is the same as the dimension of the space in which the robot is embedded. For instance, in 3D space, a point robot has three degrees of freedom.” [3]. Decomposing the problem from 3D space-time ($x \times y \times t$) which means $\mathcal{O}(n^3)$ to two times 2D, reduces the complexity to $2 \cdot \mathcal{O}(n^2)$. The objective of the trajectory planning is to move a robot from an initial point $(\vec{x}_I, t_I) \in \mathbb{R}^2 \times T$ to a target point $(\vec{x}_F, t_F) \in \mathbb{R}^2 \times T$ without collision with other obstacles O . Obstacles are represented as time dependent polygons that are implemented as time dependent point sets $O : T \rightarrow \mathcal{P}(\mathbb{R}^2)$. All obstacles are buffered according to the robots’ geometry (in case of a circle the radius is chosen for buffering) while all robots are shrunk and represented as time dependent points $\vec{x}_\pi : T \rightarrow \mathbb{R}^2$. We first explain the spatial path planning and, in a second step, the temporal path planning. First, the PPP addresses to find the shortest path from the robots current position $\vec{x}_I$ to a target point $\vec{x}_F$, that is where the action shall be executed ($\vec{x}_a$). The shortest path on a 2D surface that avoids collisions with static obstacles O^s is a composition of line segments that connects a subset of vertices of the obstacles. Two vertices can be connected by a line if they are *visible* to each other, i.e., the resulting line does not intersect with any obstacle. We create a weighted graph where the edge weights represent the Euclidean distance between two points. We use Dijkstra’s algorithm in order to find the shortest path. We call the resulting path π .

Next, we have to ensure that there will be no collisions with dynamic obstacles O^m while the robot moves along the path π . Assuming that there is another path π^* , e.g. from another robot, that crosses the path π , then there is a potential collision. We use the concept of forbidden regions in order to avoid the collision. A forbidden region is a spatio-temporal blocking of the path. In order to create the forbidden regions, we first

parametrize π as a function of the radian s : $\vec{x}_\pi(s)$. The location function $\vec{x}_\pi : S \rightarrow \vec{x}$ maps s to a point in (x, y) -space; with $S = [s_I, s_F]$. Figure 4 shows an example of creating forbidden regions visualized using a space-time diagram and finding a velocity profile which avoids collisions with dynamic obstacles. The path π has a length of 8 units, hence, $s_I = 0$ and $s_F = 8$. We assume there are two potential collisions which generate two forbidden regions. The first one, shaped as a triangle, appears in the spatial interval $s^* = [1, 4]$ of π and causes a temporal blocking in the interval $t^* = [2, 4]$. The second one, shaped as a rectangle, appears in the spatial interval $s^\# = [5, 7]$ of π and causes a temporal blocking in the interval $t^\# = [3, 6]$. We compute a velocity profile in order to avoid collisions. For this, we first create a graph by interconnecting all vertices of the forbidden regions with each other (Figure 4(a)). Again, this holds only for vertices that are visible to each other. Next, we connect the initial location s_I, t_I (s_I, t_I is the starting point in st -space while $\vec{x}_I$ is the corresponding starting point in xy -space) with all vertices of the forbidden regions. Assuming a maximum velocity of $v_{max} = 2$, the light gray area visualizes a velocity greater than v_{max} which is not possible. All vertices of the graph are connected by a v_{max} -edge with points located on the line $s = s_F = 8$. All these produced vertices ($s_F, t_{F,*}$) are finishing points (s_F, t_F is the finishing point in st -space while $\vec{x}_F$ is the corresponding finishing point in xy -space). In Figure 4(b) there are four possible finishing points although there are (excluding the finishing points) eight vertices. Only the vertices of the second forbidden region can be connected by a v_{max} -edge since the others do not fulfill the *visible* predicate. In the next step, the graph must be pruned. All edges that do not represent correct configurations have to be removed from the graph. A configuration is a velocity profile which consists of constant velocity segments. Those edges are: edges that indicate a velocity greater v_{max} and those that go backwards in time. Due to the pruned graph, some vertices might not be reachable anymore. They have to be pruned as well. Due to the pruning some of the finishing points have to be removed as well: in order to reach $s_F, t_{F,1}$ a velocity greater than v_{max} is required. Since this is not possible, $s_F, t_{F,1}$ is discarded. The second finishing point $s_F, t_{F,2}$ assumes a velocity of exactly v_{max} . However, the resulting velocity profile would lead to a collision with the second dynamic obstacle since the edge intersects with the forbidden region. This configuration is also not possible and is discarded. The same applies to $s_F, t_{F,3}$: there is one segment which states progress in s -direction while t remains constant, i.e., movement in zero time. This is physically not possible and is, hence, discarded. The last finishing point $s_F, t_{F,4}$ is reachable. The pruned graph may have multiple valid configurations. Using Dijkstra again, we compute the shortest path in the graph as depicted in Figure 4(c). This configuration is the velocity profile that the robot has to adopt in order to avoid collisions with dynamic obstacles.

IV. EVALUATION

This section evaluates the idea of virtualized motion: In the first scenario, two applications are programmed each of which

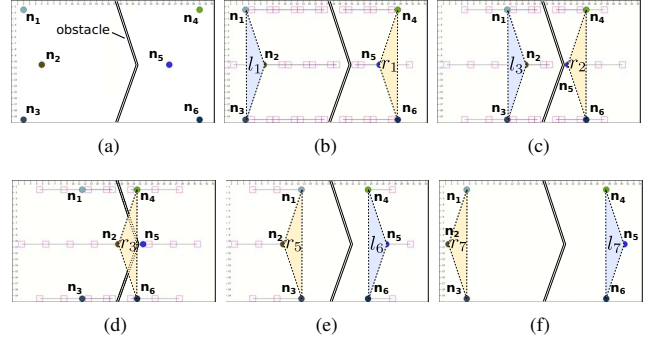


Fig. 5. Triangle movement

Suite	Action	t_{min}	t_{max}	g
as_1^l	a_1	5s	6s	(3, 1), (4, 2)
	a_2	5s	6s	(6, 10), (7, 11)
	a_3	5s	6s	(3, 19), (4, 20)
as_2^l	a_1^*	$a_1.t_{min} + 12s$	$a_1.t_{max} + 12s$	$a_1.g.x + 4$
	a_2^*	$a_2.t_{min} + 12s$	$a_2.t_{max} + 12s$	$a_2.g.x + 4$
	a_3^*	$a_3.t_{min} + 12s$	$a_3.t_{max} + 12s$	$a_3.g.x + 4$
$\vdots$				
as_7^l	...			

TABLE I
ACTION SPECIFICATIONS OF app_1

performs monitoring tasks. Similar to the example provided in Section I which has been depicted by Figure 1, each application performs three measurements at a time and then moves on. The three simultaneous measurement points take the shape of a triangle. A playground has been simulated which has a size of 33×20 units as depicted in Figure 5. Each application takes, in seven discrete steps, three measurement points resulting in a total of $2 \cdot 7 \cdot 3 = 42$ measurement points (21 per application). Each measurement point is realized by an action. The action specifications of the two applications (app_1 and app_2) are listed in Tables I and II. The first application (app_1) is implemented by virtually moving from the “left” side to the “right” while the second application (app_2) virtually moves from the “right” side to the “left”. The applications are visualized by a blue and yellow triangle. Let l_i and r_i be the triangle formation at the i -th step of app_1 and app_2 , respectively.

There are six robots available in the world ($n_1, n_2, \dots, n_6$) that are usable for application execution as depicted in Figure 5(a).

Suite	Action	t_{min}	t_{max}	g
as_1^r	a_1	11s	12s	(28, 1), (29, 2)
	a_2	11s	12s	(25, 10), (26, 11)
	a_3	11s	12s	(28, 19), (29, 20)
as_2^r	a_1^*	$a_1.t_{min} + 12s$	$a_1.t_{max} + 12s$	$a_1.g.x - 4$
	a_2^*	$a_2.t_{min} + 12s$	$a_2.t_{max} + 12s$	$a_2.g.x - 4$
	a_3^*	$a_3.t_{min} + 12s$	$a_3.t_{max} + 12s$	$a_3.g.x - 4$
$\vdots$				
as_7^r	...			

TABLE II
ACTION SPECIFICATIONS OF app_2

nodes	initial position	speed
n_1	position: (2, 1.5)	$v_{max} = 1$
n_2	position: (5, 10.5)	$v_{max} = 1$
n_3	position: (2, 19.5)	$v_{max} = 1$
n_4	position: (31, 1.5)	$v_{max} = 1$
n_5	position: (26, 10.5)	$v_{max} = 1$
n_6	position: (31, 19.5)	$v_{max} = 1$

TABLE III

WORLD SET-UP: 6 NODES FORMING TRIANGLES.

A static obstacle located in the center of the world restricts the degree of movement: robot n_1, n_2 and n_3 cannot move to the right side while robot n_4, n_5 and n_6 cannot move to the left side. Thus, executing app_1 and app_2 cannot be done by physical movement only. The initial locations of the robots are listed in Table III. Figure 5(b) visualizes the first step: the formation l_1 is shaped by the robots n_1, n_2 and n_3 while r_1 is shaped by n_4, n_5 and n_6 . Due to the actions specifications, both triangles “approach” each other as shown in Figure 5(c). The right triangle (r_2) reaches the static obstacle. In order to form r_3 , the executing set of robots needs to be changed since n_5 is not able to cross the obstacle. As a result, n_2 takes over (Figure 5(d)). In Figure 5(e) both applications have completely switched their executing robots: app_1 is now executed from robot n_4, n_5 and n_6 while app_2 is now executed from robot n_1, n_2 and n_3 . Finally, Figure 5(f) shows the 7th step: the last actions have been executed and the robots have approximately moved back to their origins. In order to execute both applications, the following physical movement is necessary: $s_{phys}^{n_1} = 26.5$, $s_{phys}^{n_2} = 30.5$, $s_{phys}^{n_3} = 26.5$, $s_{phys}^{n_4} = 19.5$, $s_{phys}^{n_5} = 13.5$ and $s_{phys}^{n_6} = 19.5$. This results in a total physical movement of 136 units. The virtual movement caused by each application is $s_{virt}(app_1) = s_{virt}(app_2) = 3 \cdot 24 + 5 = 77$ leading to a total virtual movement of $s_{virt} = 154$. The resulting efficiency is: $ef = 0.12$ which states that the physical movement was reduced by 12%. At a first glance, the benefit might not be that high, but more importantly, the scenario was made feasible by using virtualized motion. Without the virtualization and, thus, the migration, none of the applications could have been executed.

Next, we used the metrics (Section II-E) and measured the results in a simulation with random task (action) distribution. For this, we extended the world’s size to 100×100 units and placed n robots using a spatial uniform distribution in it. We generated different action sets of size 1000; each action had a duration of $d = 1$. The spatial constraints of each action was randomly generated inside the world’s borders ($(x, y) \in [0, 100] \times [0, 100]$). The temporal constraints were generated in $t_{min}, t_{max} \in [0, 100]$. All robots had a maximum velocity of $v_{max} = 1$. The initial schedule was the empty schedule ($\mathcal{S}^g = \{\}$). All experiments have been performed 100 times.

Figure 6(a) shows the number of (total) scheduled jobs as a function of the number of robots (n). The curve is “S” shaped and has two phases: in the beginning there is a superlinear increase in the number of schedulable jobs when adding more nodes followed by a degradation phase. The strong increase in the first phase appears since adding more nodes enables to reduce more physical movement and, hence, schedule

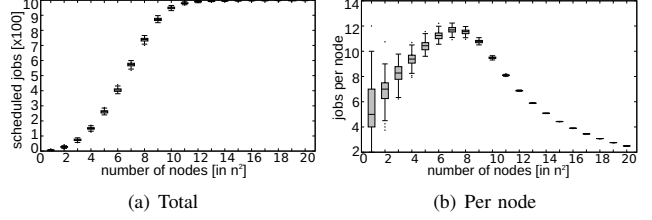


Fig. 6. Amount of scheduled jobs

superlinear more jobs. When the number of nodes reaches 64, then the degradation phase starts. At this time about 750 actions have been scheduled. Since the constraints are randomly generated, there is a certain probability that some constraints are very restrictive given by a very small time window of 1 unit (d is also 1 unit) while the space constraint might be very far away which would result in a large detour. It is even possible that an action is never schedulable, i.e., the constraints of two actions are equal and the time window is very short. In this case, when the first action gets scheduled, the second (identical) action is not schedulable since the execution location is already blocked. However, those restrictive constraints require over-proportional more robots in order to become schedulable. Since the schedule probability decreases when a certain percentage of actions have already been scheduled ($\approx 75\%$), there are over-proportional more robots required in order to achieve a 100% schedulability: the first 100 actions can be scheduled by ≈ 11 robots; the last 100 actions require ≈ 60 robots. It required ≈ 144 robots in order to schedule all actions. Figure 6(b) depicts the number of scheduled jobs per node. Using only 1 robot, there is large variance in the amount of scheduled jobs noticeable. This happens since the randomness has a large impact on the schedule order. Adding more nodes to the system, there is a strict decrease of the variance. At ≈ 100 robots, the variance is almost zero. In the first phase, there is also a strict increase in the number of actions per node. At the end of the first phase ($\approx 75\%$ of actions scheduled), the number of actions per node reaches its maximum. This is the most economic point when considering the relation of provided robots (resources) by the system to service requests (actions) by customers. Figure 7(a) plots the virtual movement as a function of the number of robots. The average “distance” between two subsequent actions is ≈ 52 . Hence, the amount of virtual movement depends linear on the amount of scheduled jobs. Scheduling all 1000 actions results in a virtual movement of 52,000 units. Figure 7(b) states the amount of physical movement that was necessary in order to execute the actions. There is also a strict increase noticeable since scheduling more actions requires more physical movement. The maximum is reached when $\approx 90\%$ of the actions have been scheduled (81 nodes necessary). However, the fraction of physical movement compared to the virtual movement is only $\approx 10\%$. Afterwards, there is a strict decrease of physical movement noticeable because adding more nodes results in even more migration and, therefore, less physical movement. Figure 7(c) shows the

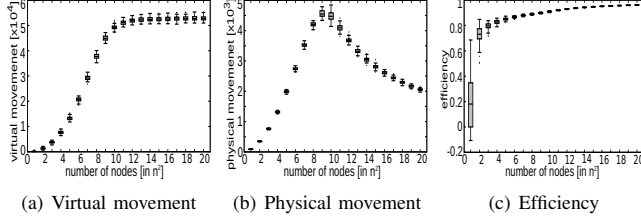


Fig. 7. Movement

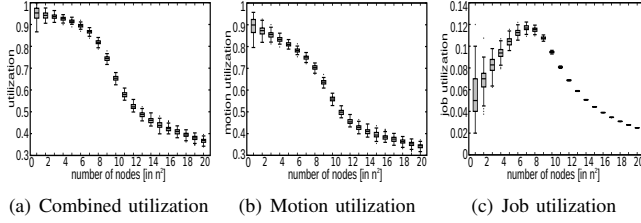


Fig. 8. Utilization

efficiency curve which states that adding more nodes to the system results in a higher efficiency. This is due to two reasons: on the one hand more actions are schedulable (increasing virtual movement) which leads to only $\approx 10\%$ more physical movement. On the other hand, if the number of schedulable actions is kept static (all actions are already scheduled) while the number of robots is increased, an improved distribution of actions among the robots is achieved such that the overall physical movement can be reduced. Adding an infinite amount of robots, the efficiency tends to approach 1. Figure 8 plots the respective utilization curves, i.e., u , u_r^m and u_r^j . The curve representing the utilization u is also “S” shaped (Figure 8(a)). In the beginning, adding more nodes to the system results in a higher percentage of scheduled jobs. Hence, the utilization remains high indicating that the system is highly utilized. After $\approx 80\%$ of the actions have been scheduled (64 nodes), the utilization decreases strictly. This occurs since adding more nodes increases the schedulability rate only slightly and leads to a highly distributed utilization. Scheduling 1000 actions using 400 nodes leads to a utilization of $u \approx 0.35$. The utilization u is the sum of the motion utilization u_r^m and the job utilization u_r^j . Figure 8(b) shows the curve for u_r^m which is approximately translated in negative y direction. Figure 8(c) depicts u_r^j . The job load increases up to 49 nodes; this is also the point at which the most jobs per node are scheduled. After this point, the schedulability rate increases only slightly and finally reaches its maximum. Therefore, the scheduled jobs are also highly distributed among the nodes leading to monotonic decrease of u_r^j . Using approximately 100 nodes or more leads to a decrease in the variance, i.e., tends to 0. This means that the schedule probability of all 1000 actions using 100 nodes or more is (almost) 1. The reason for the asymptotic behavior is that there is a certain probability that actions are not schedulable due to their constraints as already explained. In the following, we

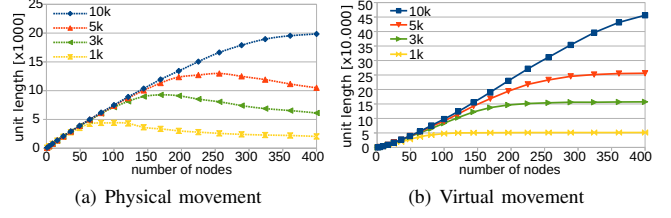


Fig. 9. Total movement

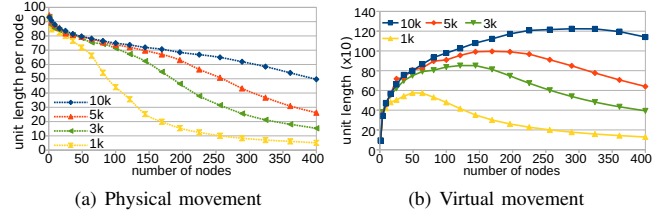


Fig. 10. Movement per node

modified the simulation set-up by scheduling action sets of size $3k$, $5k$ and $10k$ in addition. Figure 9 shows the comparison of total physical and virtual movement. The curve progressions of the physical movement (Figure 9(a)) for the respective actions sets is similar to the aforementioned $1k$ set which is given by a monotonic increase in the beginning, i.e., when more scheduled actions lead to more physical movement. At a given point (when all actions already have been scheduled), the physical movement goes down (high spatial node distribution increases probability of performing even more migration). Figure 9(b) shows the virtual movement which also shows similar curve progressions, i.e., the virtual movement increases monotonically. Comparing virtual and physical movement shows an approximate correlation of $s_{virt} \approx 20 \cdot s_{phys}$ stating a reduction of physical movement realized by migration. Figure 10 depicts the physical and virtual movement per node for the four action sets. The curve progressions for the $3k$, $5k$ and $10k$ sets are also similar to the $1k$ one. Since the virtual and physical movement is shown per node, the correlation between both is the same: $s_{virt}^{node} \approx 20 \cdot s_{phys}^{node}$. Figure 11 states the different utilizations u , u_r^m and u_r^j for the four action sets which also show the similarity of the curve progressions. The results of the evaluation have demonstrated the advantage of virtualized movement which enable to reduce the overall physical movement, increase the systems efficiency and, thus, save energy. Keeping the amount of robots constant enables to schedule more actions. Keeping the amount of actions constant enables to schedule them with fewer robots. Both ways increase the efficiency.

V. RELATED WORK

The presented approach combines conceptual ideas taken from path and motion planning for robots over vehicle routing to virtualization. Path and motion planning plays an important role in research for mobile robots. In [4], an analysis of

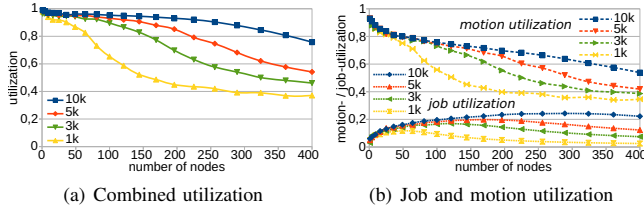


Fig. 11. Utilization with 1k, 3k, 5k and 10k actions

dynamic environments and corridor planning is presented. In most cases, environments are dynamic. The author examines different approaches for known, partially known and unpredictable dynamic environments. In [5] an approach for reactive path planning in dynamic environments is presented. In [6], a combination of a kinematic visibility graph path planning algorithm, an efficient continuous curvature path generation method based on β -Spline curves and a speed profile generator based on cubic Splines is presented. In [7], a fast scheduling algorithm called *Tercio* is presented that assigns tasks with spatio-temporal constraints to agents. The authors use a task sequencer that computes a schedule for multi-agent systems in polynomial time. The algorithm can satisfy upper and lower bound temporal deadlines as well as spatial restrictions. In [8], the *Law Enforcement Problem* (LEP) is presented. This problem shows policemen on their patrol that have to react to incidents (tasks) that also have to be addressed by multiple policemen. They present the *FMC_TA* algorithm that is based on a heuristic which best utilizes the capacity of the team for the LEP. Both approaches (*Tercio* as well as *FMC_TA*) also schedule spatio-temporal tasks in space and time. There are several revisions of the original approach presented in [3]: In [9] the prioritized planning scheme is proposed that introduces robot priorities. [10] combines priorities with the potential field method for conflict resolution. A decentralized approach that also uses priorities is presented in [11]. In [12] the approach is extended by introducing alternative routes from a start point to a target point that are adjacent paths. It considers multiple spatial paths and, thus, avoids the problem of the single spatial path approach. The vehicle routing problem (VRP) describes the problem of creating routes for vehicles that need to service a set of locations. It is a generalization of the traveling salesman problem (TSP). There are different variations of the VRP, e.g., the VRP with time windows (VRPTW) [13], the stochastic and dynamic VRPTW (SDVRPTW) [14] and the dynamic traveling repairman problem [15]. A combination of virtualization techniques adapted from cloud computing and vehicle routing is presented in [16]. The author describes a virtual vehicle that is mapped to potentially multiple physical ones applying the vehicle routing problem. Collision detection or avoidance with obstacles is not targeted in this work.

VI. CONCLUSION

In this paper we have presented the use of virtual movement and its gains for Cyber-physical systems. Virtual movement

is realized by migration of application parts from one robot to another, instead of moving one robot along the whole path specified by the application programmer. In our system, application programmers only need to state which action, e.g. measure temperature or take a picture, should be carried out where. The mapping of actions to robots is done automatically by the system. This enables automatic usage of virtual movement which can reduce the physical movement. As the motors used for the movement of robots usually have the highest influence on energy consumption, virtual movement increases the lifetime of the system.

REFERENCES

- [1] D. Graff, D. Röhrig, and R. Karnapke, "On the Need of Systemic Support for Spatio-Temporal Programming of Mobile Robot Swarms," in *11th IEEE International Conference on Mobile Ad-hoc and Sensor Networks (MSN 2015)*. IEEE Computer Society, December 2015, pp. 227–235. [Online]. Available: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7420948>
- [2] D. Graff and R. Karnapke, "Cyber-Physical Systems—Exemplary Applications and a Distributed Execution Platform," in *2016 IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies*. IEEE, August 2016, pp. 120–125.
- [3] K. Kant and S. W. Zucker, "Toward Efficient Trajectory Planning: The Path-Velocity Decomposition," *The International Journal of Robotics Research*, vol. 5, no. 3, pp. 72–89, September 1986.
- [4] J. P. van den Berg, "Path planning in dynamic environments," Ph.D. dissertation, Utrecht University, 2007. [Online]. Available: <http://arl.cs.utah.edu/pubs/PhDThesis.pdf>
- [5] F. Belkhouche, "Reactive path planning in a dynamic environment," *IEEE Transactions on Robotics*, vol. 25, no. 4, pp. 902–911, Aug 2009.
- [6] V. F. Muñoz, A. García-Cerezo, and A. Cruz, "A mobile robots trajectory planning approach under motion restrictions," *Integr. Comput.-Aided Eng.*, vol. 6, no. 4, pp. 331–348, Dec. 1999. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1275759.1275760>
- [7] M. Gombolay, R. Wilcox, and J. Shah, "Fast scheduling of multi-robot teams with temporospatial constraints," in *Proceedings of Robotics: Science and Systems*, Berlin, Germany, June 2013.
- [8] S. Amador, S. Okamoto, and R. Zivan, "Dynamic Multi-agent Task Allocation with Spatial and Temporal Constraints," in *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, ser. AAMAS '14. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2014, pp. 1495–1496. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2615731.2616029>
- [9] M. Erdmann and T. Lozano-Perez, "On multiple moving objects," *Algorithmica*, vol. 2, pp. 1419–1424, 1986. [Online]. Available: <http://lis.csail.mit.edu/pubs/tlp/multiple-moving.pdf>
- [10] C. W. Warren, "Multiple Robot Path Coordination Using Artificial Potential Fields," in *IEEE International Conference on Robotics and Automation*, Cincinnati, OH (USA), 1990, pp. 500–505.
- [11] R. Alami, F. Robert, F. Ingrand, and S. Suzuki, "Multi-Robot Cooperation through Incremental Plan-Merging," in *IEEE International Conference on Robotics and Automation*. IEEE Computer Society, pp. 2573–2579.
- [12] T. Fraichard and C. Laugier, "Path-Velocity Decomposition Revisited and Applied to Dynamic Trajectory Planning," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Atlanta, GA (USA), May 1993, pp. 40–45.
- [13] N. A. El-Sherbeny, "Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods," *Journal of King Saud University - Science*, vol. 22, no. 3, pp. 123 – 131, 2010. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S1018364710000297>
- [14] M. Pavone, N. Bisnik, E. Frazzoli, and V. Isler, "A stochastic and dynamic vehicle routing problem with time windows and customer impatience," *Mobile Networks and Applications*, vol. 14, no. 3, pp. 350–364, 2008. [Online]. Available: <http://dx.doi.org/10.1007/s11036-008-0101-1>
- [15] S. Itani, E. Frazzoli, and M. A. Dahleh, "Dynamic traveling repairperson problem for dynamic systems," in *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, Dec 2008, pp. 465–470.
- [16] J. Huang, "From the real vehicle to the virtual vehicle," Ph.D. dissertation, 2013. [Online]. Available: <http://escholarship.org/uc/item/8gb263gz>