2016 IEEE 41st Conference on Local Computer Networks

Spatio-Temporal Coordination of Mobile Robot Swarms

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

***Abstract*—Context-aware applications that require access to physical space and time are a necessity in cyber-physical systems. We focus on the design of a cyber-physical operating system in which a space-time scheduler is the core-component responsible for resource management. Given a set of space-time tasks and a set of mobile robots that move through physical space, a main objective remains in finding a mapping of tasks to robots. In this paper, we address the problem of scheduling a set of tasks with spatio-temporal constraints in space and time. We present an online-scheduler that computes collision-free spatio-temporal trajectories for the robots in order to execute the space-time tasks. As side condition, collisions with static as well as dynamic obstacles must be avoided at all times. The scheduler consists of two components: a *job scheduler* that uses a heuristic and performs a coarse-grained scheduling and a *trajectory planner* that takes the output of the job scheduler and computes spatio-temporal trajectories.**

I. Introduction

Access to physical space and time is a necessity in cyber-physical systems. We enable the development of applications in which actions (which are interactions with the physical world) can be defined and annotated with spatio-temporal constraints. Actions are executed distributed and concurrently. In [1], we described the programming model of such applications as well as the architecture of the operating system.

We provide application programmers with a programming interface that supports transactions for a set of actions. Application programmers benefit from our approach since error-prone aspects in distributed systems such as synchronization, coordination and distribution are hidden beyond the system's interface. In order to support transactions in a distributed system of mobile robots where space and time are essential side conditions, we use a distributed transaction voting that has been presented in [2].

In this paper, we describe the process of mapping individual applications, more precisely actions, to certain robots. The paper is structured as follows: Section II describes the job scheduler followed by Section III explaining the trajectory planner. In Section IV, an evaluation of the algorithm is given, Section V lists related work in this field and Section VI concludes the paper.

II. Job Scheduling

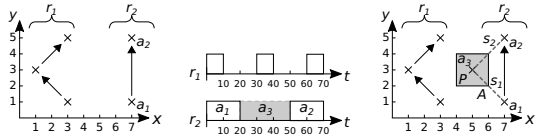
The world has 3 dimensions: 2 space dimension (surface) and a time dimension. All entities (in the world) have their own geometry represented as a 2D polygon. Entities can be either static or dynamic: the former ones stay at a fixed location while the latter ones move over time. A function $f : T \rightarrow \mathbb{R}^2$ 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 $R \backslash \{r_i\}$ are considered as dynamic obstacles O^m. The objective of the scheduler is to compute a schedule that maps actions $(a_1, a_2, .., a_n)$—created from applications—to robots under consideration of spatio-temporal constraints of $a_1, a_2, .., a_n$, which define the boundary of execution in space and time. An action $a \in A$ is defined as a tuple (g, t_{min}, t_{max}, d) with g stating the space and $[t_{min}, t_{max}]$ stating the temporal constraint while d is the duration. The scheduler computes for each action a 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, the scheduler computes a collision-free spatio-temporal trajectory in order to move r to p. Initially, the local schedule of each robot is the empty schedule: $\mathcal{S} = \{\}$. During system execution, the scheduler assigns new actions to robots such that $\mathcal{S} \neq \{\}$. Gaps in-between action jobs mark available (free) slots (movement jobs omitted). If the scheduler assigns an action to a robot, the path that the robot has to move in addition is defined as detour since it produces additional effort for the robot.

Figure 1 shows an example with two robots r_1 and r_2. The initial situation is depicted in Figure 1(a): r_1 already has three scheduled jobs and r_2 has two scheduled jobs (a_1 and a_2). The arrows indicate the movement of the respective robot and, therefore, also the local order of action execution, i.e., local means with respect to the robot. Figure 1(b) shows the local schedule of each robot; for simplicity, all movement jobs have been omitted here and, instead, only free slots are shown.

Now, a new action (a_3) shall be scheduled and need to be assigned to one of the robots. The scheduler can arbitrarily modify or remove robot trajectories as long as the robot is not yet moving along it. Hence, the local order of execution can be influenced afterwards. Already scheduled actions are, instead, committed; they are considered as fixed.

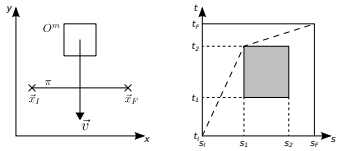
Considering the time interval $[0, 70]$ in Figure 1(b), there are three free slots: $r_1 \rightarrow [10, 30], [40, 60]$, $r_2 \rightarrow [20, 50]$. The scheduler checks if the new action (a_3) can be assigned to robot r_2 by placing a_3 in the free slot between a_1 and a_2. The free slot is defined by the interval $[t_1, t_2]$; $t_1 = 20, t_2 =$

DOI 10.1109/LCN.2016.81



(a) Initial situation. (b) Local schedule $\mathcal{S}_1$, $\mathcal{S}_2$ of r_1, r_2. (c) Schedule a_3.

Fig. 1. Schedule new action (a_3).



(a) O^m crosses π. (b) Forbidden region.

Fig. 2. Computing forbidden regions.

50. The execution location of job a_i is defined by $\vec{x}_{a_i}$. The current trajectory must be adapted in such a way that the robot does not move straight from $\vec{x}_{a_1} \rightarrow \vec{x}_{a_2}$. The new trajectory that considers a_3 is then from $\vec{x}_{a_1} \rightarrow \vec{x}_{a_3} \rightarrow \vec{x}_{a_2}$ as depicted in Figure 1(c). For performance reasons, the job scheduler is based on a heuristic, i.e., all obstacles are neglected in this step (static and dynamic) and it is assumed that the robot is able to move directly from $\vec{x}_{a_i}$ to $\vec{x}_{a_j}$ (obstacles are considered in the second step). Thus, the distance between two execution locations $\vec{x}_{a_i}$ and $\vec{x}_{a_j}$ is calculated by the Euclidean distance: $s = ||\vec{x}_{a_i} - \vec{x}_{a_j}||$. In order to schedule the new action a_3, the scheduler computes all possible slots from all robots that match the temporal constraints of the action.

Finally, the scheduler returns a set of valid slots. The slot that causes minimal additional movement (minimal detour) is selected. The detour is defined as the way the robot has to move in addition:

$$\Delta s := (s_1 + s_2) - s_{12} \tag{1}$$

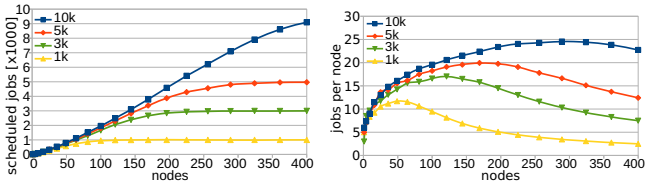
The original distance between $\vec{x}_{a_1}$ and $\vec{x}_{a_2}$ is given by s_{12} and Δs states the additional way. In case of an empty schedule, s_{12} and s_2 are both zero and, thus, $\Delta s = s_1$ holds.

Since the job scheduler is based on a heuristic, the schedule might not be feasible. Thus, the trajectory planner uses the proposed slot and checks feasibility by considering static as well as dynamic obstacles and computes a collision-free spatio-temporal trajectory. If a_3 could not be scheduled under the current variable assignment, the scheduler discards the current slot and checks the next one. If none of the slots can be taken, the scheduler picks another location point and repeats the procedure.

III. Trajectory Planning

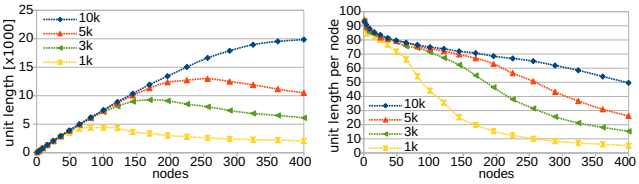
The trajectory planning is based on the path-velocity-decomposition which has been presented in [3]. The resulting problem is a trajectory planning problem (TPP). The TPP is split into the static path planning problem (PPP) and the dynamic velocity planning problem (VPP) (cf. [3]). This approach is adjusted in order to check and guarantee schedulability of a job by calculating a collision-free spatio-temporal trajectory. 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 $\vec{x}_\pi : T \rightarrow \mathbb{R}^2$. The spatial path planning computes a continuous path $\pi \subset \mathbb{R}^2$ that starts at $\vec{x}_I$ and ends at $\vec{x}_F$, where π is the spatial trajectory that the robot has to follow; π is represented as point set. This is the shortest path connecting $\vec{x}_I$ and $\vec{x}_F$ which avoids collisions with static obstacles O^s. The temporal path planning computes a temporal trajectory that states at what point in time the robot has to move along the segments of the spatial trajectory. Therefore, π is parametrized as a function of the radian measure s of the path: $\vec{x}_\pi(s)$. The location function $\vec{x}_\pi : S \rightarrow \vec{x}$ maps s to a point in (x, y)-space that is on the path π. The task is to find a time-based mapping $s : T \rightarrow S$ for s with $S = [s_I, s_F]$ being the radian interval of the path π. The spatio-temporal trajectory is then defined by $\vec{x}_\pi(t) = (\vec{x}_\pi \circ s)(t)$. The goal of the temporal planning is to find a mapping $\vec{x}_\pi : T \rightarrow \pi$ between time t and the points of the path π, so that collisions with dynamic obstacles are avoided. For the explanation of the trajectory planning, the example provided in Figure 1 is used. In order to check schedulability, the trajectory planner uses the proposed path segments (s_1, s_2) and computes a collision-free trajectory for each of them (considering static and dynamic obstacles).

First, the algorithm computes two spatial paths π_1 and π_2 where π_1 is the path from $\vec{x}_{a_1}$ to $\vec{x}_{a_3}$ (approximated by the line segment s_1 as shown in Figure 1(c)) and π_2 is the path from $\vec{x}_{a_3}$ to $\vec{x}_{a_2}$ (approximated by the line segment s_2). Second, the temporal paths σ_1 and σ_2 for π_1 and π_2 are calculated. Finally, π_1 and σ_1 are combined to the trajectory τ_1. The same applies to π_2 and σ_2 which are combined to τ_2. Figure 2 depicts the computation of τ. Figure 2(a) shows the path π of a robot that moves on a straight line from $\vec{x}_I$ to $\vec{x}_F$. A dynamic obstacle O^m moves downwards possibly resulting in a collision where the path π and the extended velocity vector $\vec{v}$ intersect. In order to avoid this situation, a velocity profile (σ) is calculated. For each potential intersection, a forbidden region is created which spans a space-time area in which the collision might occur. Figure 2(b) shows a translation of the intersection of π and $\vec{v}$ (in (x, y)-space) as a forbidden region in (s, t)-space. In order to determine σ, we use graph theory and find the shortest path in the graph that finally represents the desired velocity profile (depicted as dashed line). The interpretation is that we first decrease the velocity and allow the obstacle to pass. Afterwards, we increase the velocity in order to be in time at the desired location. The algorithm calculates the latest possible start time: scheduling the movement job (trajectory) as



(a) Total scheduled jobs (b) Scheduled jobs per node

Fig. 3. Scheduled jobs



(a) Total physical movement (b) Movement per node

Fig. 4. Physical movement

late as possible increases the probability that the robot is able to execute more tasks than when scheduling movement jobs as soon as possible. This is due to the fact that movement jobs can be arbitrarily replaced as long as the robot is not yet moving along the trajectory.

IV. Evaluation

In order to evaluate the scheduling algorithm, we iteratively scheduled four different *action-sets* of size *1k*, *3k*, *5k* and *10k* onto a set of robots. Each action set is scheduled with an initial empty global schedule ($\mathcal{S}^g = \{\}$). We simulated a 2-dimensional world of size 100×100 spatial units. We examined the amount of schedulable actions as a function of the number of robots. All robots have a maximum velocity of $v_{max} = 1$ and the duration of actions is set to 1 second.

The actions of each set are scheduled sequentially. Each action has spatio-temporal constraints that were randomly generated: spatial constraints are generated inside the worlds border, i.e., $(x, y) \in [0, 100] \times [0, 100]$ and are equally-distributed in the spatial world; temporal constraints are generated in $t_{min}, t_{max} \in [0, 100]$.

Figure 3(a) shows the total amount of schedulable actions as a function of the number of nodes. Each of the four curves represents one action set. All of the curves are "S" shaped. In the beginning, there is a superlinear increase of scheduled jobs when adding more nodes. This is due to the fact that with each additional node in the system, the average physical distance to a potential new action is reduced. Not all actions of the sets are schedulable because of resource insufficiencies, e.g., omitting all space constraints and assuming each action requires 1 second for execution, one robot could, at maximum, execute 100 actions. Since there are spatial constraints, the amount of scheduled actions is less (approximately 5). In average the robot had to move approximately 19 space units for each action. Assuming $v_{max} = 1$, the robot moved $5 \cdot 19 = 95$ units which took 95 seconds. The overall execution took 5 seconds which reflects a full utilization of the robot. After a certain amount of actions have been scheduled (from the respective action sets), the slope of the curves degrades and slowly approaches their maximum. This happens since the amount of actions in each set is fixed. Since the constraints are randomly generated, there might be overlapping constraints or constraints that are hard to fulfill. Hence, the curves approach their maximum slowly. Figure 3(b) shows the amount of scheduled jobs per node. Scaling up the number of nodes in the system allows to schedule superlinear more actions and, thus, increases the system efficiency. There is a superlinear increase in the beginning, i.e., each node is able to execute more jobs. Each of the curves have a maximum followed by a degradation phase. The beginning of the degradation phase (Figure 3(a)) corresponds approximately to the maximum of jobs per node (Figure 3(b)), e.g., for the $1k$ set, the degradation phase starts at 50 nodes which is also the maximum of jobs per node (≈ 12). It requires ≈ 121 (≈ 361) nodes to schedule all $1k$ ($5k$) actions. Each node executes ≈ 8 (≈ 14) actions. This shows the increase of efficiency when scaling up the number of nodes.

Figure 4(a) shows the total physical movement necessary in order to execute the scheduled actions. There are again 3 phases: an initial increase, a maximum and a degradation phase. The initial increase results from more nodes that are added to the system and all of them being highly utilized, i.e., the scheduler requires the added nodes in order to schedule the actions. This leads to higher total movement. Focusing on the $1k$ curve, all actions can be scheduled using 121 nodes. This is also approximately the maximum of the total physical movement (≈ 5000 space units). Adding more nodes to the system does not lead to a higher schedulability since all $1k$ jobs are already scheduled. However, the average physical movement decreases, which leads to less total movement. Figure 4(b) depicts the average physical movement per node. The curves clearly state that, in general, adding more nodes to the system leads to a strict decrease in the average physical movement per node which enables to schedule more jobs per node.

Next, we analyze the system utilization. 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

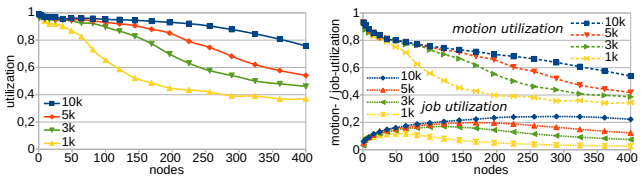
$$u(t_1, t_2) := \frac{u_1(t_1, t_2) + u_2(t_1, t_2) + .. + u_n(t_1, t_2)}{n} \quad (2)$$

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 $u_r^m(t_1, t_2)$:

$$u_r(t_1, t_2) := u_r^j(t_1, t_2) + u_r^m(t_1, t_2) \quad (3)$$

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

$$u_r^j(t_1, t_2) := \frac{\sum_{j \in J_r \mid j \in [t_1, t_2]} dur(j)}{\Delta_t} \quad (4)$$



(a) Utilization (b) Motion and job utilization

Fig. 5. System utilization

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

$$u_r^m(t_1, t_2) := \frac{\sum_{m \in M_r \mid v(m)>0} dur(m)}{\Delta_t} \tag{5}$$

M_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 $dur(m)$ delivers the time duration of the trajectory length for all segments which have a velocity greater than 0 ($v(m) > 0$).

Figure 5(a) shows the (combined) utilization u while Figure 5(b) depicts the motion u_r^m and job utilization u_r^j. As already stated, the average physical movement strictly decreases when adding more nodes to the system. This is confirmed by the motion utilization curves, i.e., they strictly decrease. Due to the decrease, the average node has more free capacity and is able to execute more actions. This is indicated by the increasing job utilization curves. The curves also have three phases: increase, maximum and degradation. This is due to the fixed amount of actions per action set, i.e., u_r^j becomes smaller when all actions of the set have already been successfully scheduled and the amount of nodes is further increased. In this case, the average amount of jobs per node becomes less which leads to a lower job utilization.

V. Related Work

In [4], 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 in polynomial time for multi-agent systems. They are able to satisfy upper and lower bound temporal deadlines as well as spatial restrictions. In [5], the *Law Enforcement Problem* (LEP) is presented. This problem shows policemen on their patrol who 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. The main difference to our algorithm is that both approaches are based on a task model that clearly states the execution location. Furthermore, none of the approaches computes spatio-temporal trajectories. Our approach is based on the Path-Velocity-Decomposition as presented in [3] that decomposes the problem into the PPP and the 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 $2 \cdot \mathcal{O}(n^2)$. There are several revisions of the original approach presented in [3]: In [6], the prioritized planning scheme is proposed that introduces robot priorities. [7] combines priorities with the potential field method for conflict resolution. A decentralized approach that also uses priorities is presented in [8]. In [9], the approach is extended by introducing alternative routes from a start point to a target point that are adjacent paths. The switching from one path to an adjacent path also allows spatial collision avoidance of dynamic obstacles and not solely temporal collision avoidance by velocity profile adjustment. It considers multiple spatial paths and, thus, avoids the problem of the single spatial path approach.

VI. Conclusion

In this paper we presented a task planning scheduler for mobile robots. It schedules a set of actions with spatio-temporal constraints at runtime and assigns the resulting jobs to the nodes. If no robot is at the desired location of execution, the scheduler also computes collision-free spatio-temporal trajectories for the robots.

References

[1] D. Graff, D. Röhrig, R. Jasper, H. Parzyjegla, and J. Rabaey, "Operating System Support for Mobile Robot Swarms," in *Second International Workshop on the Swarm at the Edge of the Cloud*, Seattle, Washington (USA), April 2015.

[2] D. Graff, D. Röhrig, and R. Karnapke, "Systemic Support for Transaction-Based Spatial-Temporal Programming of Mobile Robot Swarms," in *40th IEEE Conference on Local Computer Networks Workshops (LCN Workshops)*, Clearwater Beach, Florida (USA), October 2015.

[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] 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.

[5] 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

[6] 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

[7] 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.

[8] 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.

[9] 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.