Integrated Motion Planning, Coordination and Control for Fleets of Mobile Robots

ICAPS 2018 Tutorial

Federico Pecora, Masoumeh Mansouri

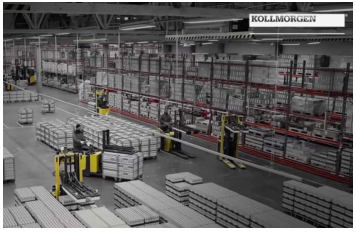
Center for Applied Autonomous Sensor Systems (AASS)
Örebro University, Sweden

SEMANTIC ROBOTS
Knowledge Foundation

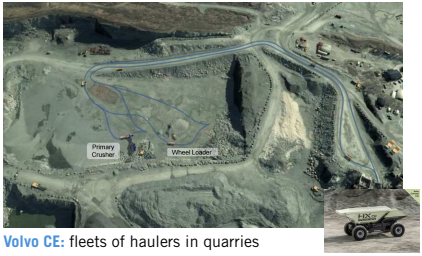
ILIAD
European Commission

Outline

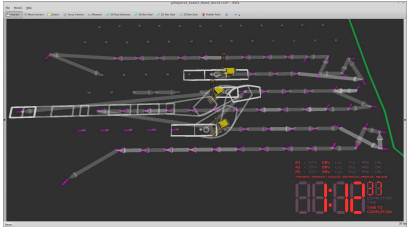
The Fleet Automation Problem [Andreasson et al., 2015]



Kollmorgen: fleets of forklifts in warehouses



Volvo CE: fleets of haulers in quarries



Epiroc / Atlas Copco: fleets of drilling machines in mines

Volvo GTO: fleets of kitting robots for production

The Fleet Automation Problem [Andreasson et al., 2015]

Mining

(e.g., Epiroc)

Construction

(e.g., Volvo CE)

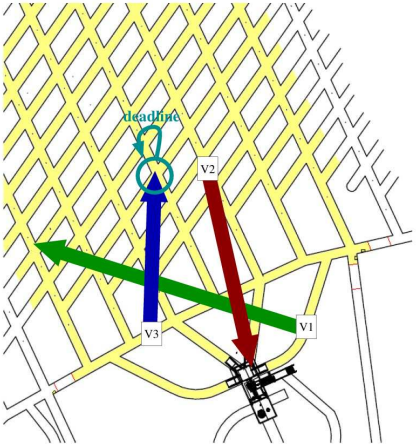
Logistics

(e.g., Kollmorgen)

Overarching aim

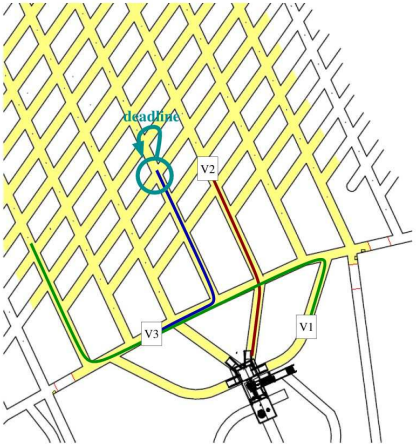
Develop general methods for fleet automation that can be applied to different industrial domains

Task Allocation in a Nutshell



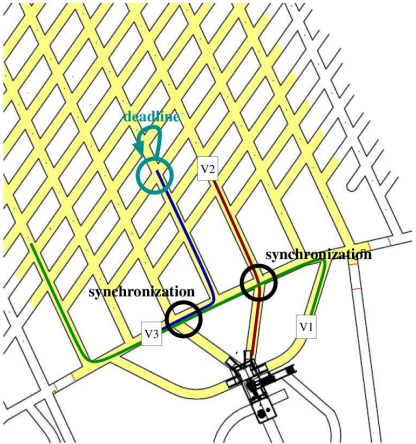
- **Task allocation:** compute where vehicles should go and when, **given**
  - drivable areas
  - tasks to be performed
  - temporal constraints (e.g., deadlines)
  - overall mission requirements (e.g., costs)

Motion Planning in a Nutshell



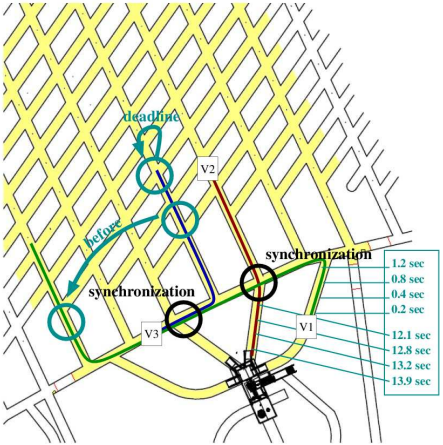
- **Motion planning:** compute vehicle paths, **given**
  - drivable areas
  - location goals
  - vehicle kinematic models

Coordination in a Nutshell



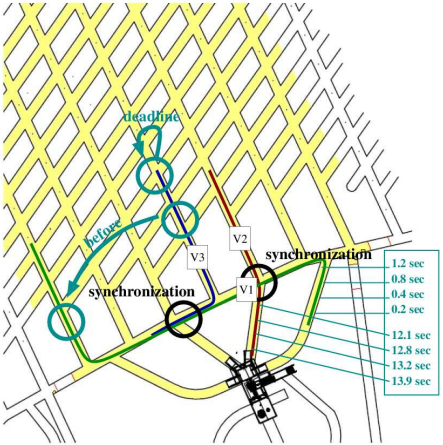
- **Coordination:** compute synchronizations that avoid collisions with other vehicles and deadlocks, **given**
  - vehicle paths
  - bounds on speed

Temporal Reasoning in a Nutshell



- **Temporal reasoning:** compute vehicle speeds/traversal times, **given**
  - synchronizations
  - vehicle paths
  - temporal constraints (e.g., deadlines and precedences)

Control in a Nutshell



- **Control:** execute vehicle motions, **given**
  - reference paths
  - reference speeds/traversal times
  - drivable area

Is The Fleet Automation Problem Hard?

- Techniques exist for solving **each sub-problem**
- But obtaining **mutually feasible** solutions is difficult

Is The Fleet Automation Problem Hard?

- Techniques exist for solving **each sub-problem**
- But obtaining **mutually feasible** solutions is difficult
- The search space is huge!

$$\{ \text{possible goal assignments} \} \times \{ \text{possible paths} \} \times \{ \text{possible synchronizations} \} \times$$
$$\{ \text{possible speeds} \} \times \{ \text{possible control actions} \}$$

Is The Fleet Automation Problem Hard?

- Techniques exist for solving **each sub-problem**
- But obtaining **mutually feasible** solutions is difficult
- The search space is huge!

$$\{ \text{possible goal assignments} \} \times \{ \text{possible paths} \} \times \{ \text{possible synchronizations} \} \times \{ \text{possible speeds} \} \times \{ \text{possible control actions} \}$$

- Focus of this tutorial
  - A Meta-CSP based approach for multi-robot coordination
    (trajectory envelope representation, systematic approach to coordination)

Is The Fleet Automation Problem Hard?

- Techniques exist for solving **each sub-problem**
- But obtaining **mutually feasible** solutions is difficult
- The search space is huge!

$$\{\text{ possible goal assignments }\} \times \{\text{ possible paths }\} \times \{\text{ possible synchronizations }\} \times$$
$$\{\text{ possible speeds }\} \times \{\text{ possible control actions }\}$$

- Focus of this tutorial
  - A Meta-CSP based approach for multi-robot coordination
    (trajectory envelope representation, systematic approach to coordination)
  - Integrating task allocation, motion planning and coordination
    (use of local search and hybrid heuristics, real world application)

Is The Fleet Automation Problem Hard?

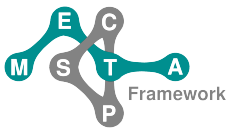
- Techniques exist for solving **each sub-problem**
- But obtaining **mutually feasible** solutions is difficult
- The search space is huge!

$$\{ \text{possible goal assignments} \} \times \{ \text{possible paths} \} \times \{ \text{possible synchronizations} \} \times \{ \text{possible speeds} \} \times \{ \text{possible control actions} \}$$

- Focus of this tutorial
  - A Meta-CSP based approach for multi-robot coordination
    (trajectory envelope representation, systematic approach to coordination)
  - Integrating task allocation, motion planning and coordination
    (use of local search and hybrid heuristics, real world application)
  - Loosely-coupled multi-robot coordination, motion planning and control
    (accounting for robot dynamics, online goal posting, scalability)

Tools and Libraries Used in This Tutorial

- We will mention examples implemented with the following libraries



metacsp.org

coordination_oru

github.com/FedericoPecora/coordination_oru

Getting the tutorial examples

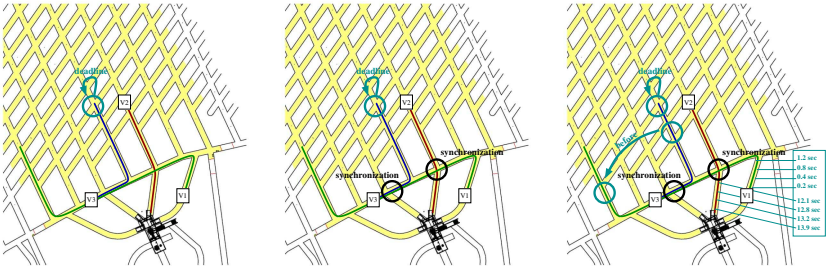
```
$ git clone https://gitsvn-nt.oru.se/fopa/coordination-tutorial-src-ICAPS-2018.git
$ cd coordination-tutorial-src-ICAPS-2018
$ ./gradlew eclipse
```

Then, import project in Eclipse

Outline

Scheduling-Based Muti-Robot Coordination

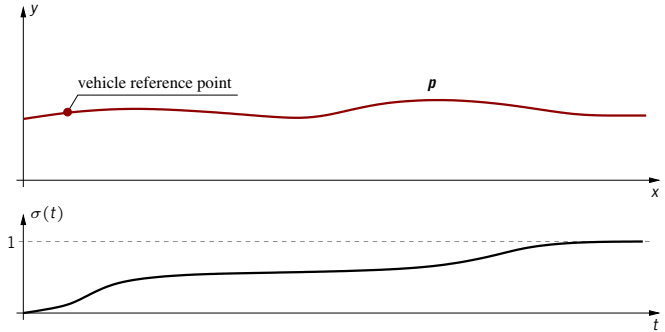
- **Idea:** impose **increasingly tight constraints** on trajectories



- Constraints **exclude** trajectories that are **kinematically infeasible** and/or lead to **collisions/deadlocks**
- Choose specific trajectories **at execution time**, revise online **when needed**

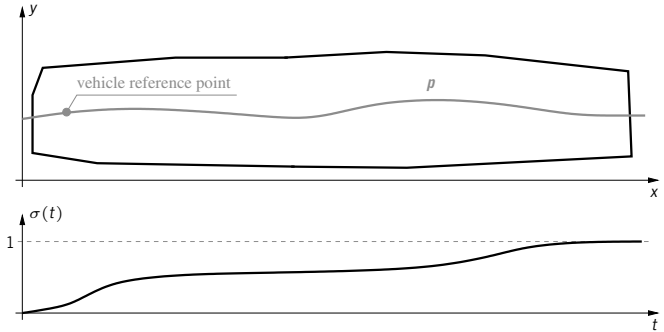
Representing Trajectories as Constraints [Pecora et al., 2012]

- **Trajectory:** $\boldsymbol{p}(\sigma(t))$ (σ = time history along path)



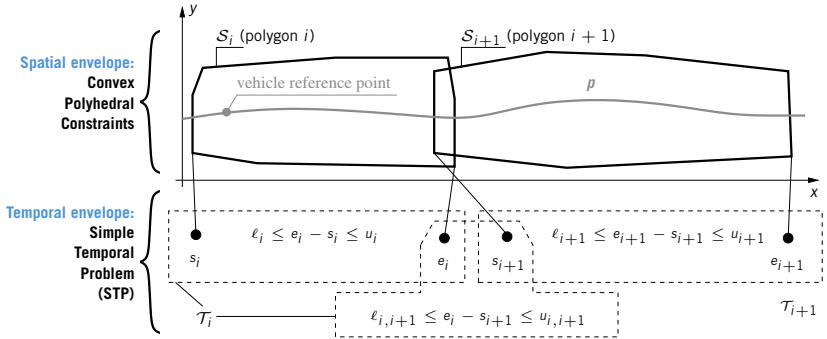
Representing Trajectories as Constraints [Pecora et al., 2012]

- **Trajectory:** $\boldsymbol{p}(\sigma(t))$ (σ = time history along path)

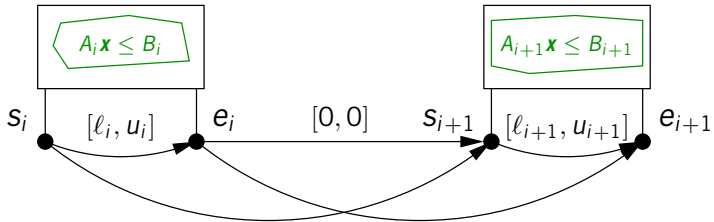


Representing Trajectories as Constraints [Pecora et al., 2012]

- **Trajectory:** $\boldsymbol{p}(\sigma(t))$ (σ = time history along path)
- **Trajectory envelope:** spatial constraints on $\boldsymbol{p}$, temporal constraints on σ



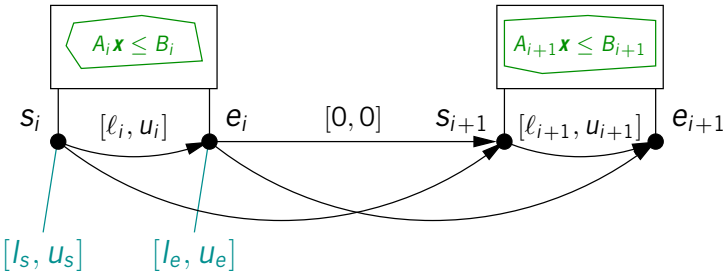
Spatial and Temporal Constraints



- Bounds computed from **slowest** and **fastest** speed traversal of reference path

$$\ell_i = e_i^{\text{fast}} - s_i^{\text{fast}}, \quad u_i = e_i^{\text{slow}} - s_i^{\text{slow}}$$

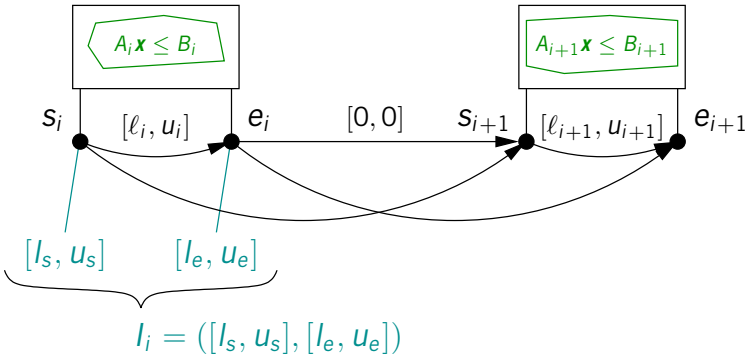
Spatial and Temporal Constraints



- Bounds computed from **slowest** and **fastest** speed traversal of reference path

$$\ell_i = e_i^{\text{fast}} - s_i^{\text{fast}}, \quad u_i = e_i^{\text{slow}} - s_i^{\text{slow}}$$

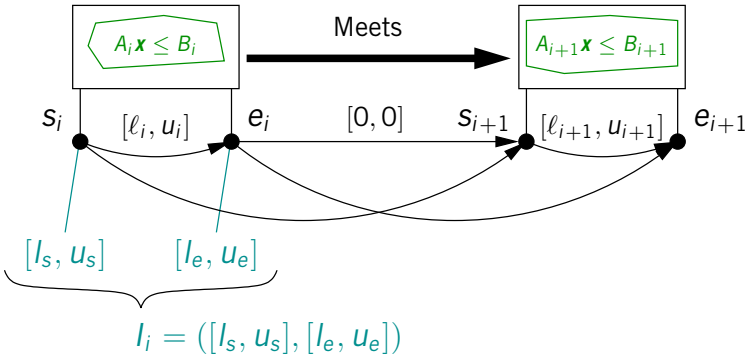
Spatial and Temporal Constraints



- Bounds computed from **slowest** and **fastest** speed traversal of reference path

$$\ell_i = e_i^{\text{fast}} - s_i^{\text{fast}}, \quad u_i = e_i^{\text{slow}} - s_i^{\text{slow}}$$

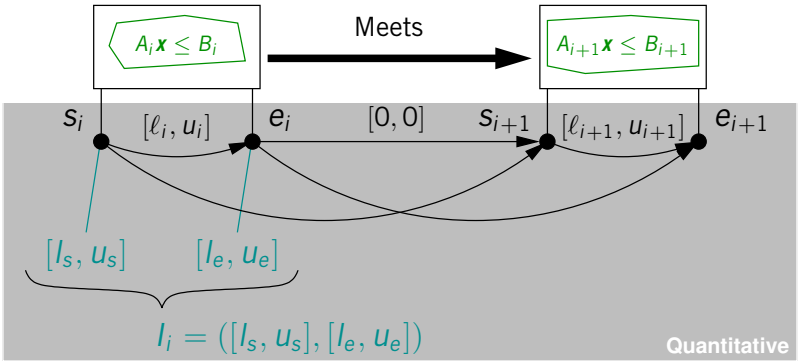
Spatial and Temporal Constraints



- Bounds computed from **slowest** and **fastest** speed traversal of reference path

$$\ell_i = e_i^{\text{fast}} - s_i^{\text{fast}}, \quad u_i = e_i^{\text{slow}} - s_i^{\text{slow}}$$

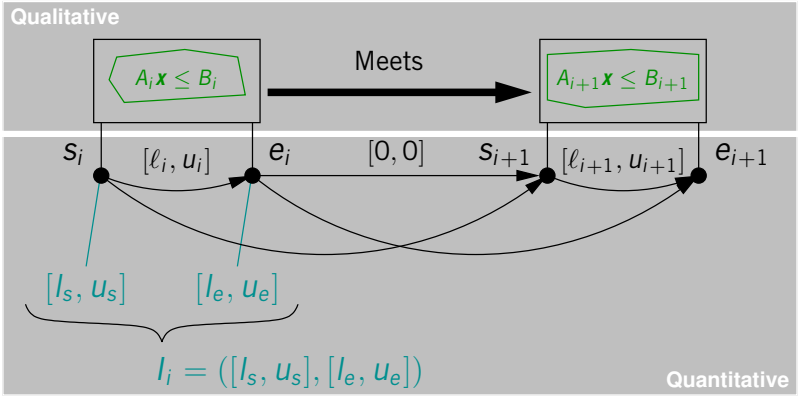
Spatial and Temporal Constraints



- Bounds computed from **slowest** and **fastest** speed traversal of reference path

$$\ell_i = e_i^{\text{fast}} - s_i^{\text{fast}}, \quad u_i = e_i^{\text{slow}} - s_i^{\text{slow}}$$

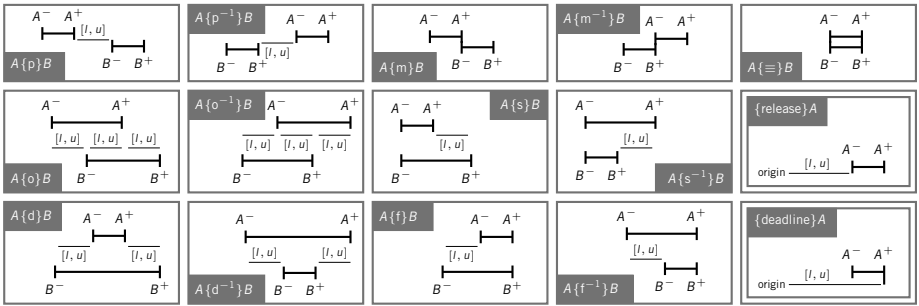
Spatial and Temporal Constraints



- Bounds computed from **slowest** and **fastest** speed traversal of reference path

$$\ell_i = e_i^{\text{fast}} - s_i^{\text{fast}}, \quad u_i = e_i^{\text{slow}} - s_i^{\text{slow}}$$

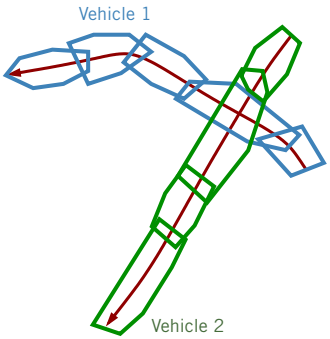
Qualitative Temporal Constraints: Augmented Allen Interval Relations



- Each qualitative relation has precise **metric semantics**
- Qualitative relations can be augmented with **bounds**

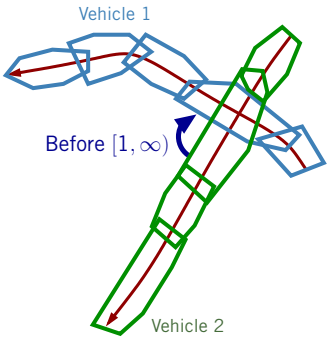
Avoiding Collisions with Temporal Constraints

- **Collision:** polyhedra of **different vehicles** that overlap in both **time** and **space**



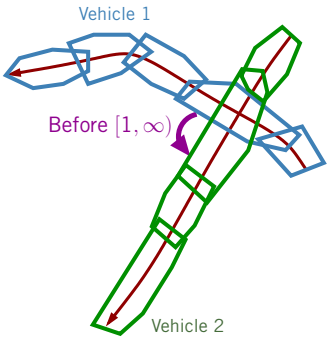
Avoiding Collisions with Temporal Constraints

- **Collision:** polyhedra of **different vehicles** that overlap in both **time** and **space**
- Collisions can be eliminated by **temporal constraints**



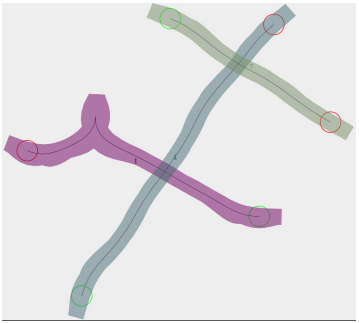
Avoiding Collisions with Temporal Constraints

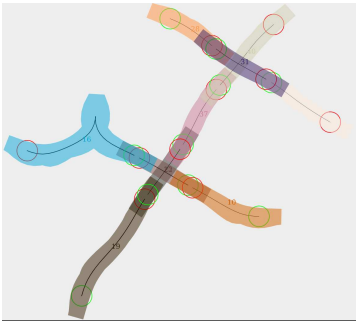
- **Collision:** polyhedra of **different vehicles** that overlap in both **time** and **space**
- Collisions can be eliminated by **temporal constraints**



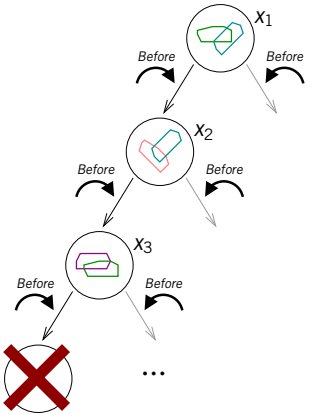
Trajectory Envelope Refinement

- We need a **granular** spatio-temporal representation
- Trajectory envelopes can be divided **around regions where they intersect** (spatially), or with **fixed strategy**





Search Space of Temporal Resolvers



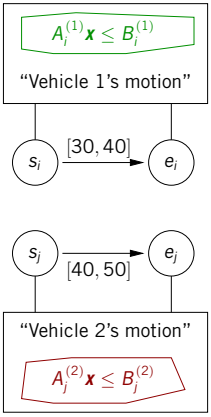
- **Variables:** spatio-temporally intersecting trajectory envelopes
- **Values:** temporal constraints
- Search space can be explored via **backtracking search**
- Heuristics can exploit both **temporal and spatial information** to guide search

Exploring the Search Space of Temporal Resolvers

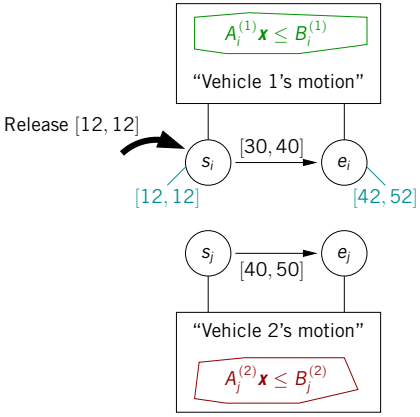
Algorithm 1: coordinate(envelopes, constraints): success or failure

```
vars ← get-spatio-temporally-overlapping-envelopes(envelopes, constraints)
if vars ≠ ∅ then
    var ← choose-variable(vars)
    values ← get-values(var)
    while values ≠ ∅ do
        value ← choose-value(values)
        if (CN ∪ value) is consistent then
            result ← coordinate(CN ∪ value)
            if result ≠ failure then
                return result
        values ← values \ value
    return failure
return success
```

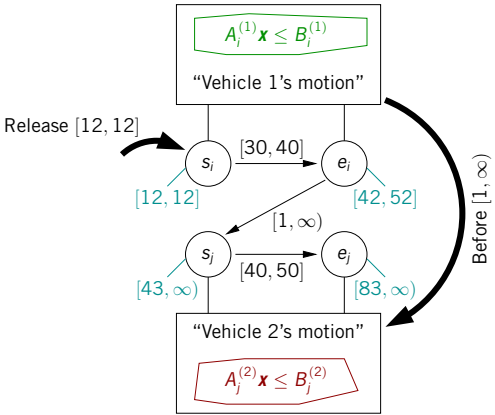
Coordinating Motion with Constraints



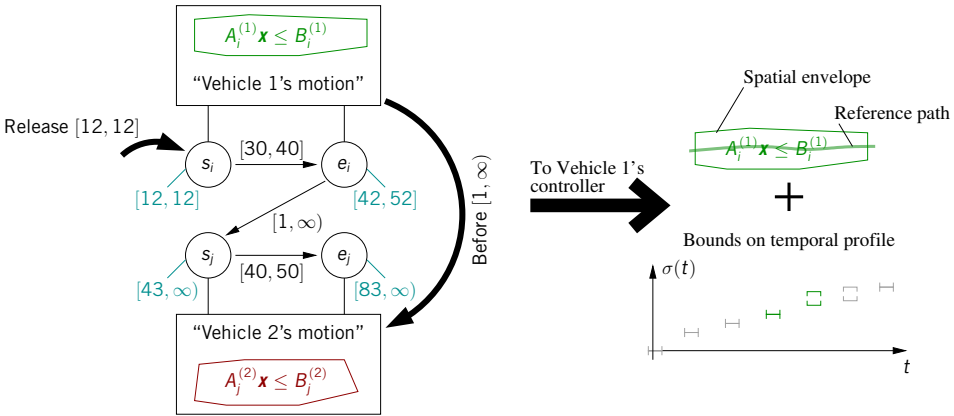
Coordinating Motion with Constraints



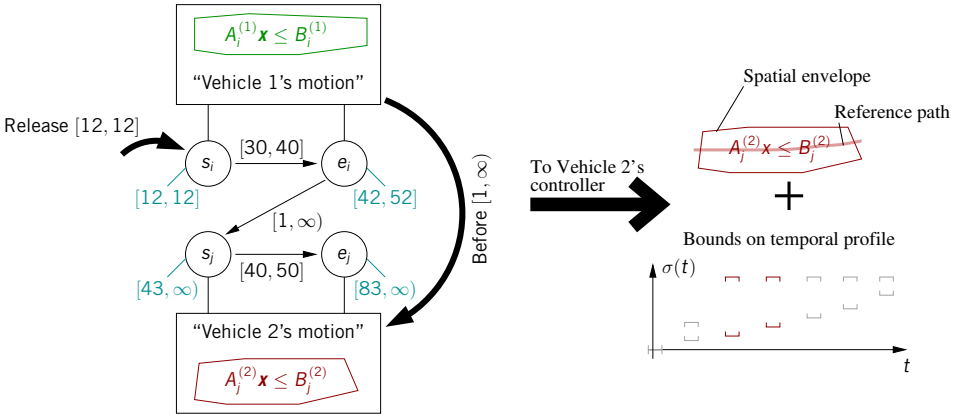
Coordinating Motion with Constraints



Coordinating Motion with Constraints

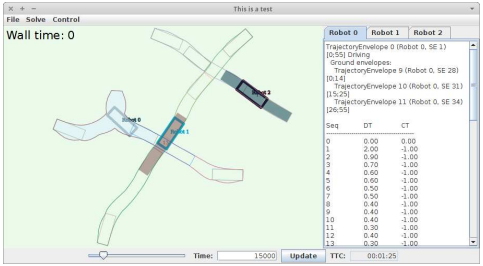


Coordinating Motion with Constraints



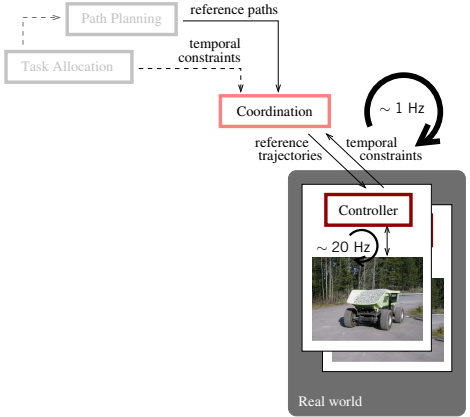
Example: Scheduling Trajectory Envelopes

`TestTrajectoryEnvelopeScheduling`

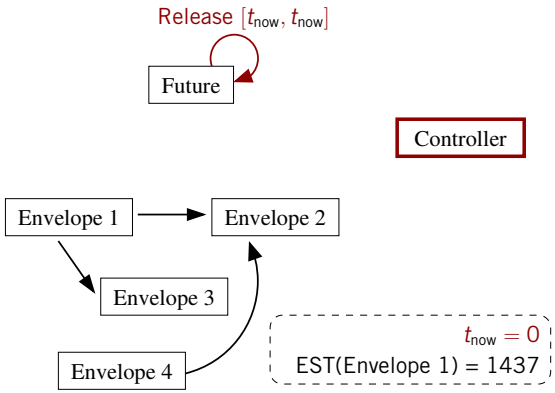


- Visualization of **trajectory envelopes** and underlying **constraint network**
- **Refine** envelopes and **solve** scheduling problem manually
- Note the **temporal constriants** on trajectory envelopes
- **Constraints among envelopes of different vehicles** are the result of scheduling

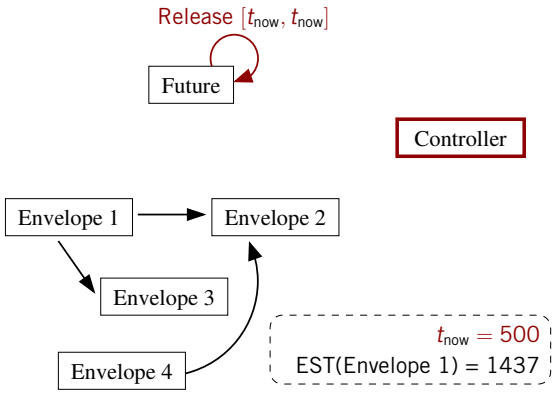
Closing the Loop with Execution: Coordination + Control



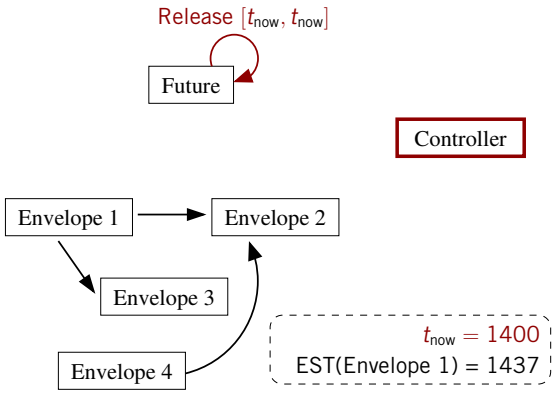
Executon Monitoring, Communication with Controllers



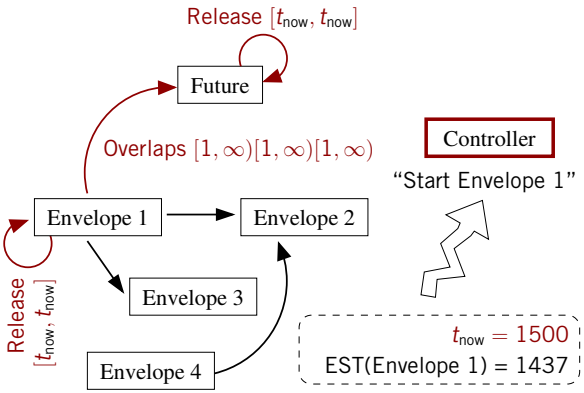
Executon Monitoring, Communication with Controllers



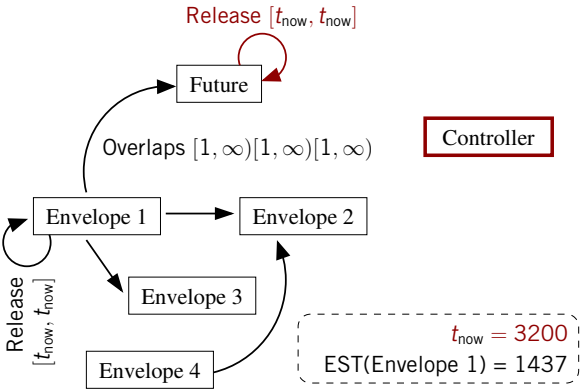
Executon Monitoring, Communication with Controllers



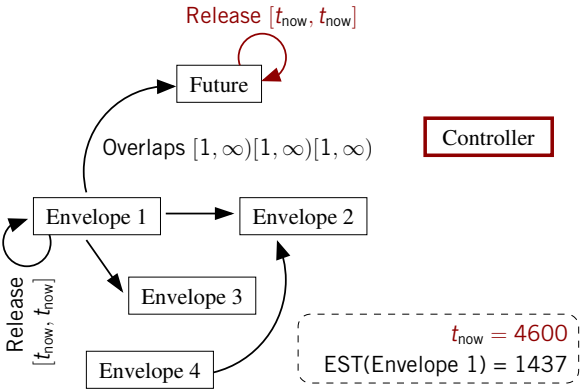
Executon Monitoring, Communication with Controllers



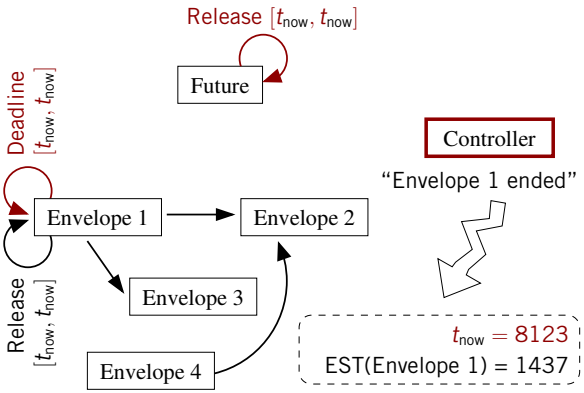
Executon Monitoring, Communication with Controllers



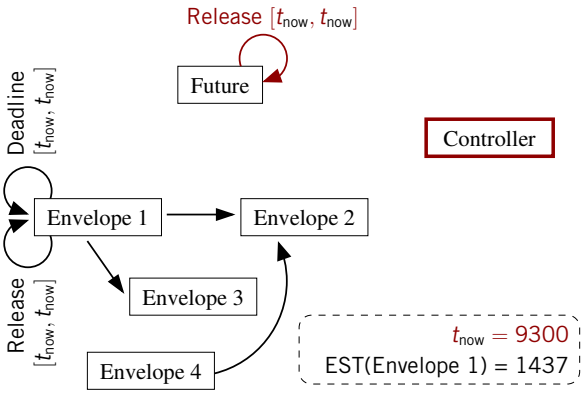
Executon Monitoring, Communication with Controllers



Executon Monitoring, Communication with Controllers



Executon Monitoring, Communication with Controllers



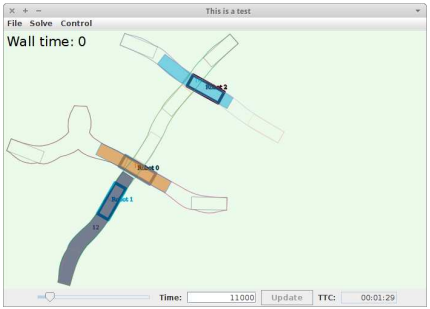
Closing the Loop with Execution: Coordination + Control

Overall, we have implemented a **high-level control loop**

1. Sample state of all robots
   ⇝ *impose appropriate constraints reflecting end of envelopes*
2. For each idle robot, compute trajectory envelope to new goal
   ⇝ *refine envelopes*
3. Find all spatio-temporal intersections
4. Compute resolving precedence constraints
   ⇝ *the* `coordinate` *algorithm*
5. Update trajectories of all robots, sleep, goto 1
   ⇝ *extract temporal profiles from STP, send to controllers*

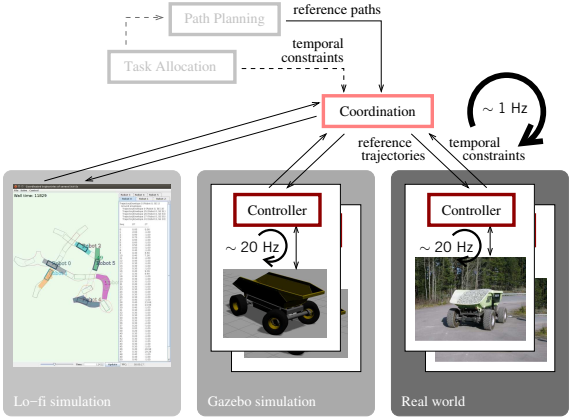
Example: Closing the Loop with Execution

`TestTrajectoryEnvelopeControl`



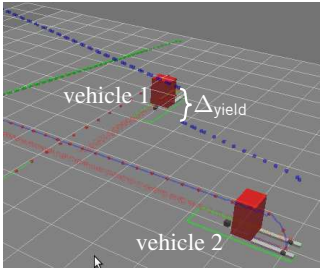
- Visualization of **scheduling** and **execution** of trajectory envelopes
- Envelopes are **dispatched** when EST occurs via a `DispatchingFunction`
- Each `DispatchingFunction` is an “ideal controller” which traverses envelopes in nominal time

Closing the Loop with Execution: Coordination + Control

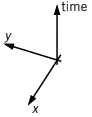


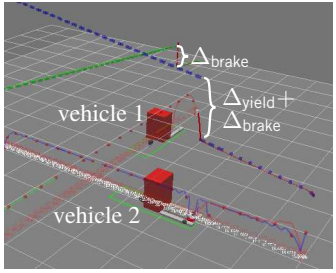
Constraints at Work: Coordinator/Controller Interaction

- Controllers report progress in current envelope as **deadline** updates
- STP solver **propagates consequences**
- Controllers are sent trajectory updates **online**



vehicle 1 will yield to vehicle 2



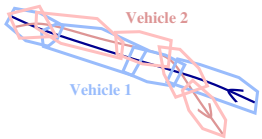


delay of vehicle 1 is propagated

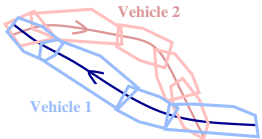
Coordination Algorithm on Real Robots

Tightly Integrated Motion Planning & Coordination [Cirillo et al., 2014]

- **Theorem:** when coordinator succeeds, solutions are **deadlock-free**
- However, **independently planned motions** may lead to unsolvable situations



Independently planned motions



Jointly planned motions

- **Solution:** plan motion for *N* vehicles **jointly when necessary**

Formal Properties

Lemma (Completeness of Motion Planning)

Given a multi-robot problem where parallel motions are not required, the motion planner is complete.

Lemma (Schedulability of motion plans)

If the motion planner finds a joint plan from C_s to C_g, the plan is guaranteed to be schedulable.

Lemma (Correctness of Coordination)

The `ScheduleTrajectories()` *algorithm is correct.*

Lemma (Completeness of Coordination)

The `ScheduleTrajectories()` *algorithm is complete.*

Useful Formal Properties

Theorem (Correctness)

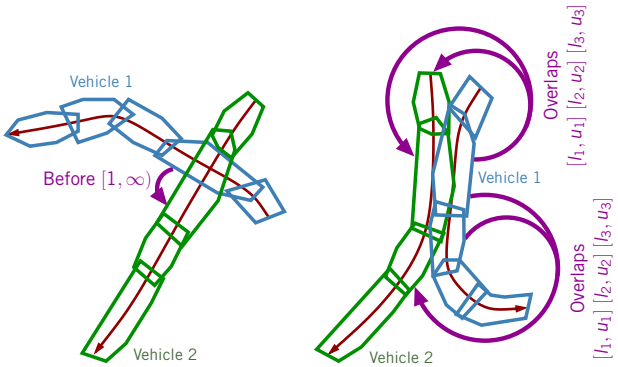
If motion planning and coordination succeed, the resulting trajectory envelopes contain at least one trajectory that is **kinematically feasible**, **conflict- and deadlock-free**.

Theorem (Completeness)

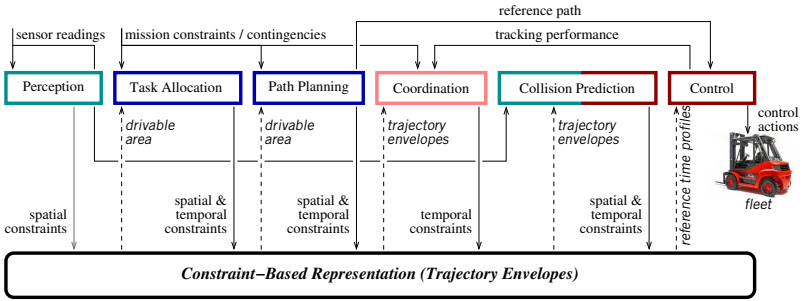
If a set of kinematically-feasible, conflict- and deadlock-free trajectories exists, the motion planner and coordinator will yield a **non-empty set of trajectory envelopes**.

Reasoning About Space and Time

- Rich representation could be exploited further...
- E.g., to reason about **direction of motion**

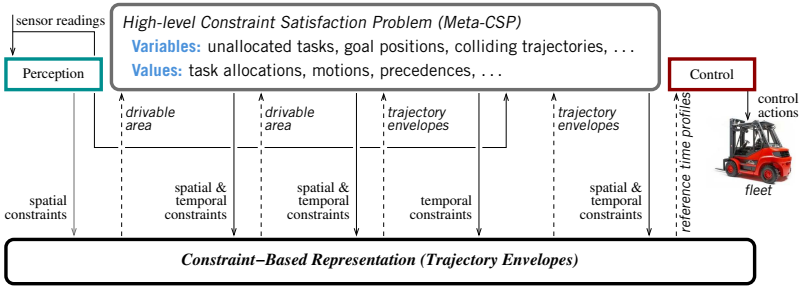


Beyond Coordination with Meta-CSP Search



- Resolving collisions and deadlocks via temporal and spatial adjustment [Mansouri et al., 2015]
- Integrated task allocation, motion planning and coordination [Mansouri et al., 2016]
- Local search for task-level reasoning + Meta-CSP search for coordination [Mansouri et al., 2017]

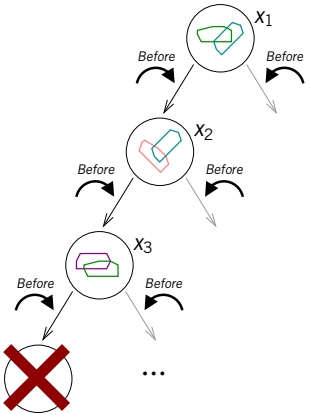
Beyond Coordination with Meta-CSP Search



- Resolving collisions and deadlocks via temporal and spatial adjustment [Mansouri et al., 2015]
- Integrated task allocation, motion planning and coordination [Mansouri et al., 2016]
- Local search for task-level reasoning + Meta-CSP search for coordination [Mansouri et al., 2017]

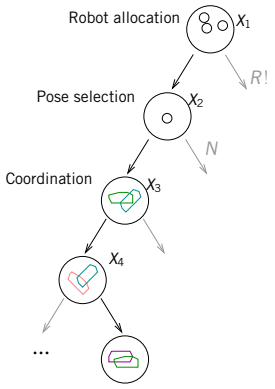
Outline

Search Space of Temporal Resolvers



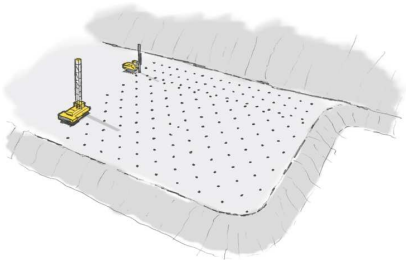
- **Variables:** spatio-temporally intersecting trajectory envelopes
- **Values:** temporal constraints
- Search space can be explored via **backtracking search**
- Heuristics can exploit both **temporal and spatial information** to guide search

Integrated Task Allocation, Motion Planning and Coordination [Mansouri et al., 2016]



- **Robot allocation:** allocating robots to the tasks
  - **Variables:** tasks that do not have an assigned robot
  - **Values:** various ways of assigning tasks to robots
- **Pose selection:** select approach angle to reach
  - **Variables:** tasks w/out an approach angle
  - **Values:** N number of discretized possible angles
- . . .

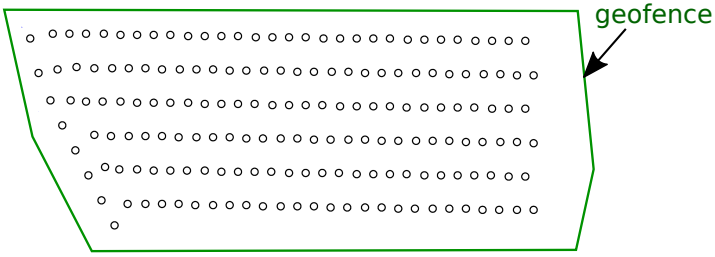
The Drill Planning Problem



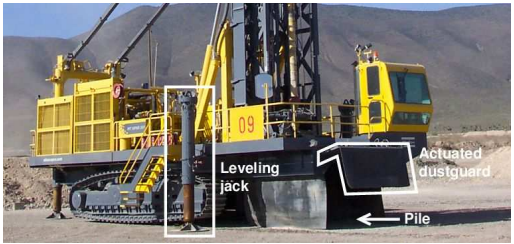
- Plan and coordinate blast-hole drilling with multiple machines
- Solutions consist of executable plans for multiple vehicles operating concurrently

Problem Requirements

- Motions should be executable by the machines (**kinematic feasibility**)
- Machines should not collide with **obstacles**, **other machines**, the **Geofence** and **drilled piles**



Problem Requirements



- Machines create **piles** after drilling
- Machines cannot drive over piles
- Piles create **obstacles** for other machines

This Problem Is Difficult to Solve!

Because, we have to decide

- which machine drills which drill target
- in which order targets are drilled
- the approach angle of each machine at each drill target
- how machines maneuver between targets
  - avoiding collisions, deadlocks and dead-ends
  - avoiding piles

. . . and should scale to hundreds of targets

This Problem Is Difficult to Solve!

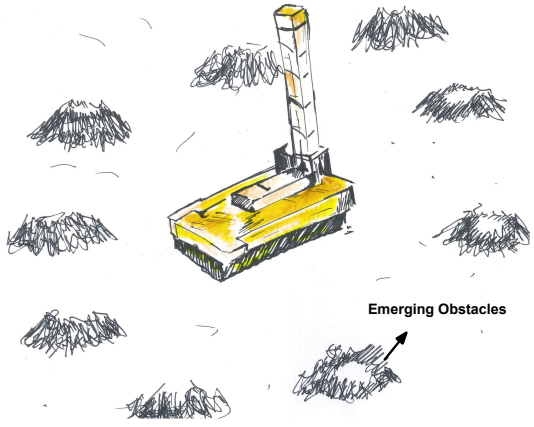
Because, we have to decide

- which machine drills which drill target
- in
- the approach angle of each machine at each drill target
- how machines maneuver between targets
  - avoiding collisions, deadlocks and dead-ends
  - avoiding piles

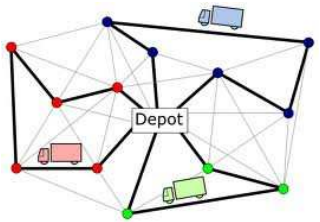
. . . and should scale to hundreds of targets

. . . and emerging piles make all these decisions strongly interdependent

Interdependencies leading to a dead-end

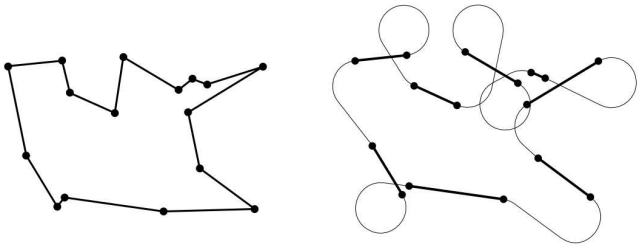


Multi Vehicle Routing Problem (MVRP) [Toth and Vigo, 2001]



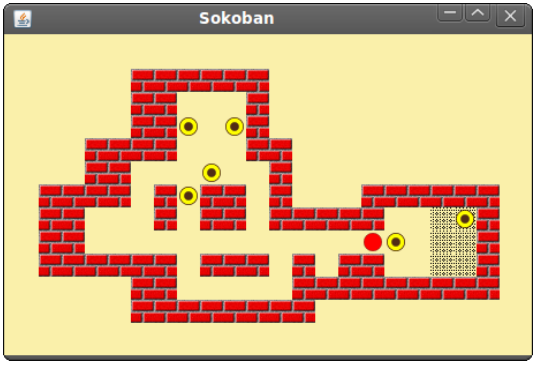
- **Not** considering **vehicle's pose** at locations, their **motions** and **dynamic obstacles**
- Solvers **ignoring** the effects of kinematic constraints and dynamic obstacles on route **cost**

Travelling Salesperson Problems for the Dubins Vehicle [Váňa and Faigl, 2015]



. . . but it does **not** consider **dynamic obstacles**

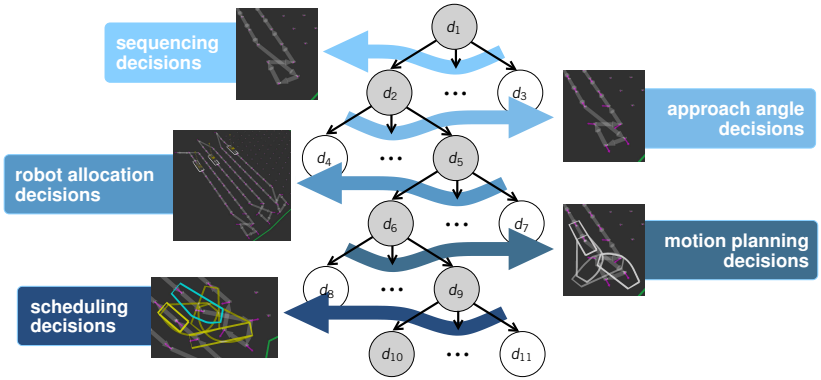
Sokoban [Culberson, 1997]



. . . “wrong” choice of actions not only affects **optimality**, but also **solvatiblity**!

Search in the Space of Sub-Problems

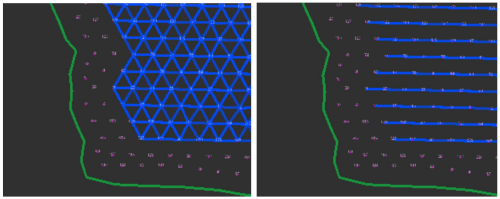
Assignments to decision variables explored via backtracking

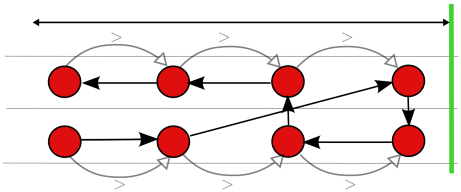


Simulation of Three Drilling Machines [Mansouri et al., 2016]

The Search Space is Huge!

Well, we use several different value-(variable) ordering heuristics.





Value-ordering heuristics for **sequencing sub-problem**

Advantages and Pitfalls of CSP-based Approach for Drill Planning

+ “Easily” integrating a new sub-problem into the overall problem
  - reducing it to a CSP
– Scales badly (up to about 100 targets)
– Not good at optimization!

MVRP-DDO as an Optimization Problem [Mansouri et al., 2017]

$$\max_{q_k \in Q} \left\{ \eta_1 len_{H_T}(q_k) + \eta_2 \sum_{(\tau_i,\tau_j) \in H_{q_k}} len_P(i,j) \right\}$$

→ Minimizing Makespan

subject to the following constraints:

$\forall q \in Q, (\tau_i, \tau_j) \in q^2$ s.t. $i \neq j$. $\tau_i \prec \tau_j \vee \tau_j \prec \tau_i$ → Sequencing Constraints

$\forall q \in Q, (\tau_i, \tau_j) \in q^2$ s.t. $(\tau_i, \tau_j) \in H_q$. $\exists P(i,j)$ → Position and Orientation Constraints

$\forall (\tau_i, \tau_j, \tau_k) \in T^3$ s.t. intersects(i,j,k) . $\tau_j \prec \tau_k$

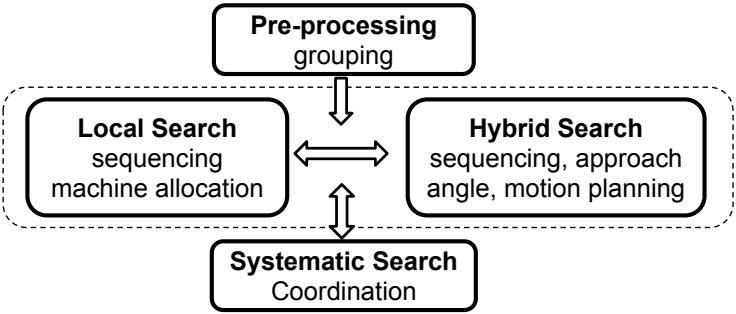
$\forall q_z, q_{w \neq z} \in Q, (\tau_i, \tau_j) \in H_{q_z}, (\tau_k, \tau_l) \in H_{q_w}$ s.t.
intersects(i,j,k,l) . $\tau_j \prec \tau_k \vee \tau_l \prec \tau_i$

→ Spatio-temporal Constraints
path-path intersection
path-pile intersection

$f(\boldsymbol{x}) = C$ → Kinematic Constraints

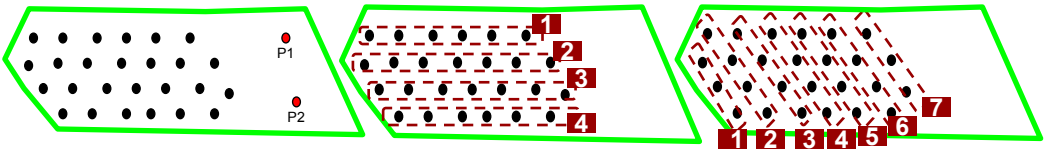
Multi Vehicle Routing Problem with Nonholonomic Constraints and Dense Dynamic Obstacles (MVRP-DDO)

Multi-abstraction Search



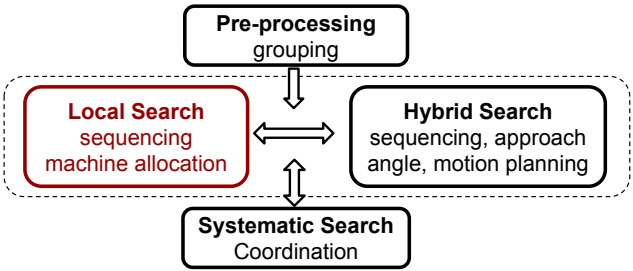
Inspired by: C. Zhang and J. A. Shah, **Co-optimizating multi-agent placement with task assignment and scheduling**, IJCAI, 2016.

Pre-processing: Grouping



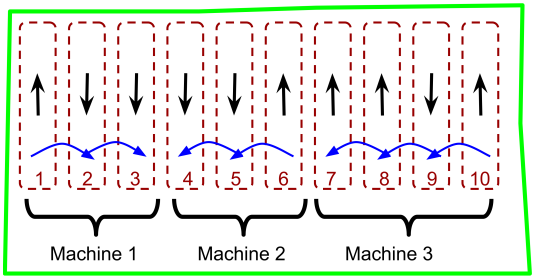
- **K-Means clustering** discovers principal directions to **divide targets into groups**
- Group is a set of drill targets that will be drilled by **one machine**

Local Search: Simulated Annealing

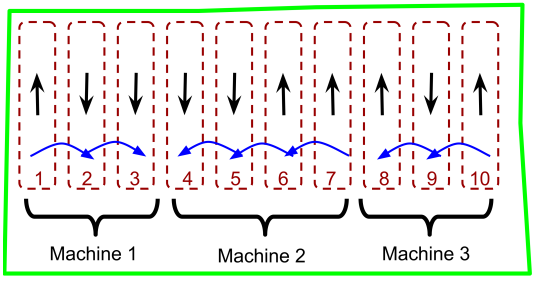


- A local search technique inspired by thermodynamics
  - start from a **complete state formulation**
  - **iteratively** modify state
  - use **cost function** to decide what to modify

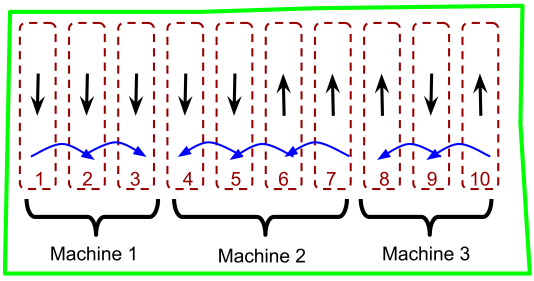
States in Simulated Annealing



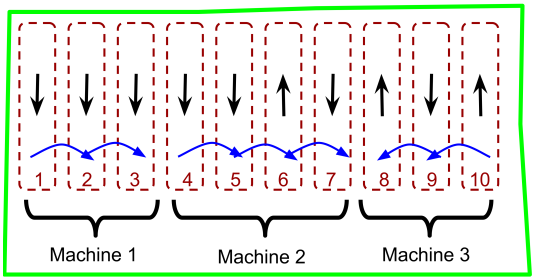
States in Simulated Annealing



States in Simulated Annealing



States in Simulated Annealing



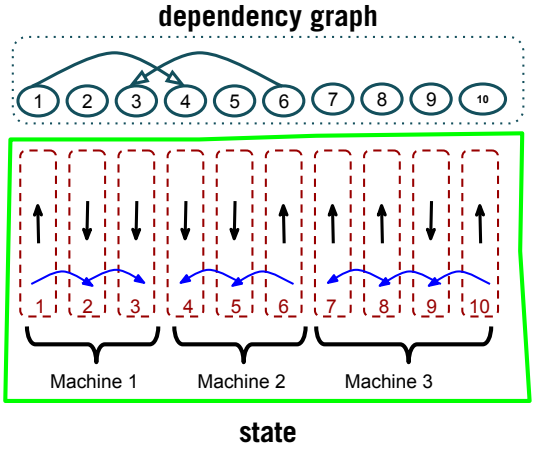
Cost Function

$$h = \max_{r \in R} TCT_r$$

- we are interested in minimizing h
- TCT_r is a total completion time of robot r
- TCT_r is computed by an approximate **multi-robot scheduler** using low fidelity simulation of robot motions
- scheduling is done at the level of **groups**

Simulation of Three Drilling Machines [Mansouri et al., 2016]

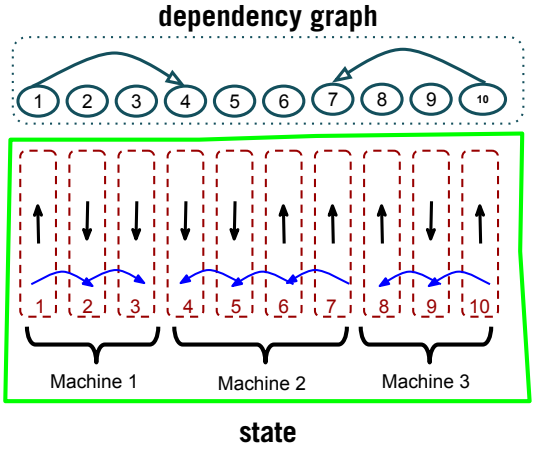
Low Fidelity Multi-robot Scheduler



Function `cost-function(s)` : makespan

```
1  G_d ← Compute(N,E)
2  if FeasibilityEval(s,G_d) then
3      return ∞
4  makespan ← 0
5  if |E| = 0 then
6      for i : n do
7          t_v = Eval(s,v_i,∅)
8          if t_v > makespan then
9              makespan ← t_v
10     return makespan
11 foreach R ∈ 𝒫(n,n) do
12     foreach v ∈ R do
13         t_v = Eval(s,v,G_d)
14         if t_v > makespan then
15             makespan ← t_v
16 return makespan
```

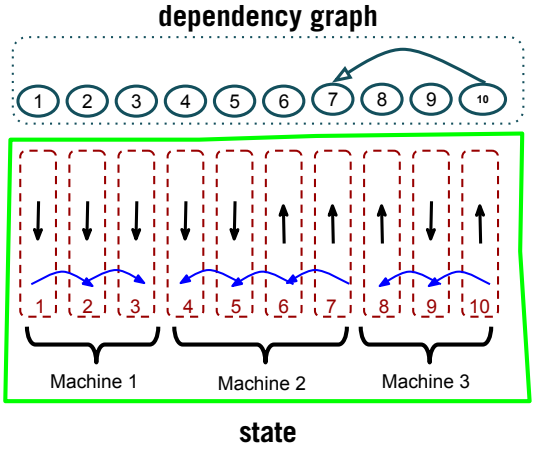
Low Fidelity Multi-robot Scheduler



Function `cost-function` (s) : makespan

```
1  G_d ← Compute(N, E)
2  if FeasibilityEval(s, G_d) then
3      return ∞
4  makespan ← 0
5  if |E| = 0 then
6      for i : n do
7          t_v = Eval(s, v_i, ∅)
8          if t_v > makespan then
9              makespan ← t_v
10     return makespan
11 foreach R ∈ P(n, n) do
12     foreach v ∈ R do
13         t_v = Eval(s, v, G_d)
14         if t_v > makespan then
15             makespan ← t_v
16 return makespan
```

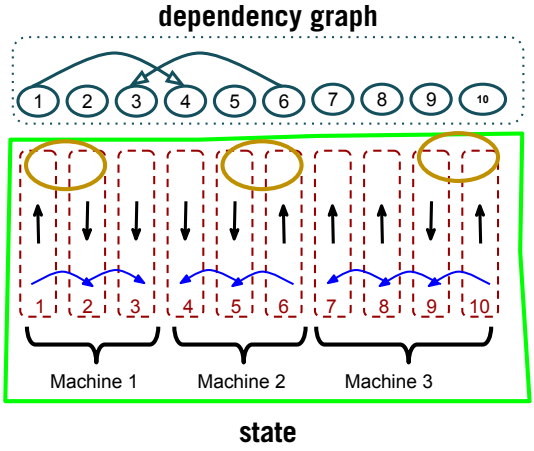
Low Fidelity Multi-robot Scheduler

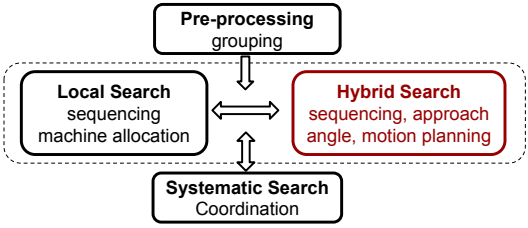


Function `cost-function` (s) : makespan

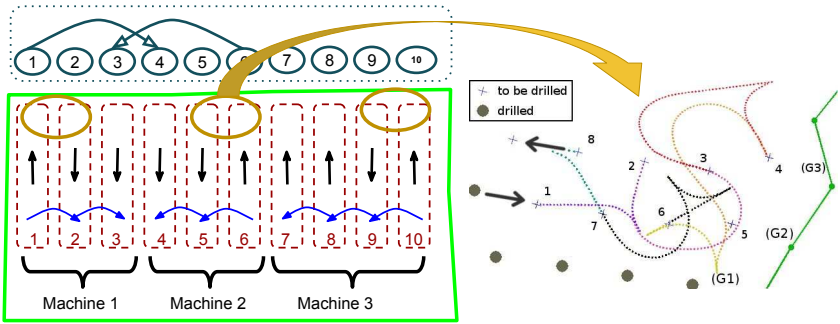
```
1  G_d ← Compute(N, E)
2  if FeasibilityEval(s, G_d) then
3      return ∞
4  makespan ← 0
5  if |E| = 0 then
6      for i : n do
7          t_v = Eval(s, v_i, ∅)
8          if t_v > makespan then
9              makespan ← t_v
10     return makespan
11 foreach R ∈ 𝒫(n, n) do
12     foreach v ∈ R do
13         t_v = Eval(s, v, G_d)
14         if t_v > makespan then
15             makespan ← t_v
16 return makespan
```

Hybrid Search





Hybrid Search

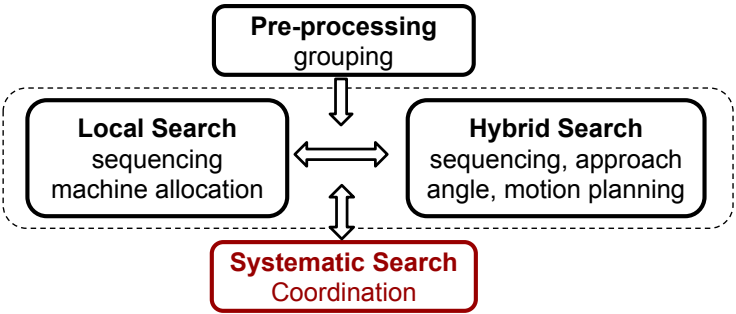


- **backward search** in the space of drill sequences
- **forward search** in the space of approach angles
  (motion planner verifies the feasibility of chosen approach angles [Liang et al., 2005])

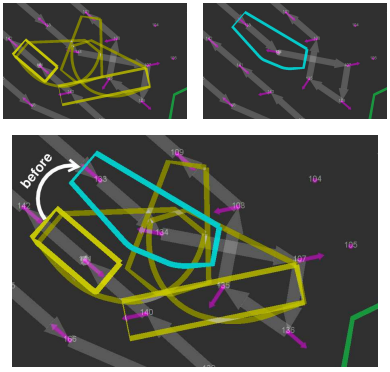
So far . . .

- we have the total sequencing for each machine
- we compute the motions of each machine so as to
  - avoid static obstacles, e.g., walls, Geofence
  - avoid the piles created by the machine itself
  - disregard **concurrent movements** of **other** machines
  - disregard **piles created** by **other** machines

Overall Architecture [Mansouri et al., 2017]



Coordination via Meta-CSP Search [Pecora et al., 2012]

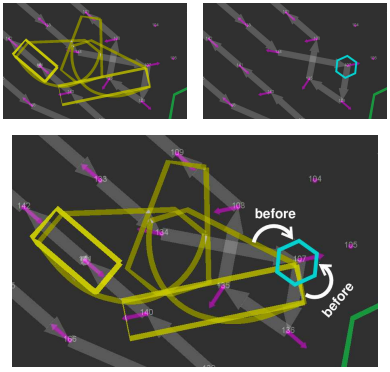


Considers spatio-temporal overlap **between machines**

So far . . .

- we have the total sequencing for each machine
- we compute the motions of each machine so as to
  - avoid static obstacles, e.g., walls, Geofence
  - avoid the piles created by the machine itself
  - disregard **concurrent movements** of **other** machines
  - disregard **piles created** by **other** machines

Coordination via Meta-CSP Search [Pecora et al., 2012]



Considers spatio-temporal overlap **between machines and piles**

Simulation of Three Drilling Machines [Mansouri et al., 2017]

Summary

- Solving multi-robot drill planning in open-pit mines
- What works best: Hierarchical approach dealing with the problem at **different levels of abstractions**
- Using different kinds of search for different sub-problems
  - **local search** for **makespan optimization**
  - combination of **backward and forward search** for “hard” part of the problem
  - systematic **backtracking** search for **multi-robot coordination**

Outline

Advantages and Pitfalls of Scheduling-Based Approach

+ Guarantees deadlock-free solutions
+ Will account for any set of temporal constraints out of the box

Advantages and Pitfalls of Scheduling-Based Approach

+ Guarantees deadlock-free solutions
+ Will account for any set of temporal constraints out of the box
– Scales badly (up to about 10 robots)

Advantages and Pitfalls of Scheduling-Based Approach

+ Guarantees deadlock-free solutions
+ Will account for any set of temporal constraints out of the box
– Scales badly (up to about 10 robots)
– Does not account for robot dynamics
  - assumes that “robots can always stop”

Advantages and Pitfalls of Scheduling-Based Approach

+ Guarantees deadlock-free solutions
+ Will account for any set of temporal constraints out of the box
– Scales badly (up to about 10 robots)
– Does not account for robot dynamics
  - assumes that “robots can always stop”
– Fleet is left in unsafe state if
  - communication breaks down
  - no deadlock-free solution exists

Advantages and Pitfalls of Scheduling-Based Approach

+ Guarantees deadlock-free solutions
+ Will account for any set of temporal constraints out of the box
– Scales badly (up to about 10 robots)
– Does not account for robot dynamics
  - assumes that “robots can always stop”
– Fleet is left in unsafe state if
  - communication breaks down
  - no deadlock-free solution exists
– Makes assumptions on robot controllers
  - which must accept temporally bounded reference trajectories,
  - and must account for updates of these bounds

Closing the Loop with Execution: Coordination + Control

Let's take another look at the **high-level control loop**...

1. Sample state of all robots
   ⇝ *impose appropriate constraints reflecting end of envelopes*
2. For each idle robot, compute trajectory envelope to new goal
   ⇝ *refine envelopes*
3. Find all spatio-temporal intersections
4. Compute resolving precedence constraints
   ⇝ *the* `coordinate` *algorithm*
5. Update trajectories of all robots, sleep, goto 1
   ⇝ *extract temporal profiles from STP, send to controllers*

Closing the Loop with Execution: Coordination + Control

Let's take another look at the **high-level control loop**...

1. Sample state of all robots
   ⇝ *impose appropriate constraints reflecting end of envelopes*
2. For each idle robot, compute trajectory envelope to new goal
   ⇝ *refine envelopes*
3. Find all spatio-temporal intersections
4. Compute resolving precedence constraints
   ⇝ *the* `coordinate` *algorithm*
5. Update trajectories of all robots, sleep, goto 1
   ⇝ *extract temporal profiles from STP, send to controllers*

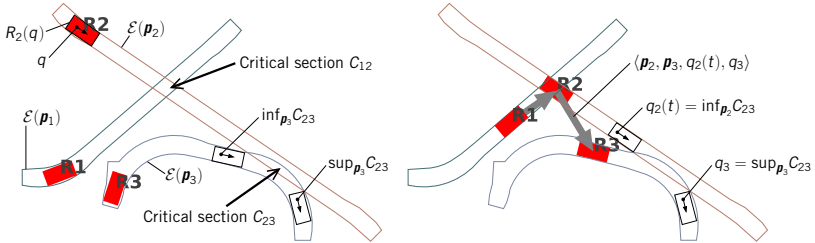
Closing the Loop with Execution: Coordination + Control

Streamline **precedence computation** and **communication to/from controllers**

1. Sample state of all robots
   ⇝ *impose appropriate constraints reflecting end of envelopes*
2. For each idle robot, compute trajectory envelope to new goal
   ⇝ *refine envelopes*
3. Update set of critical sections
4. Compute resolving precedence constraints
   ⇝ *infer precedences while accounting for heuristic and robot dynamics*
5. Update critical points of all robots, sleep, goto 1
   ⇝ *tell controllers last allowed configuration they can reach*

Critical Points [Pecora et al., 2018]

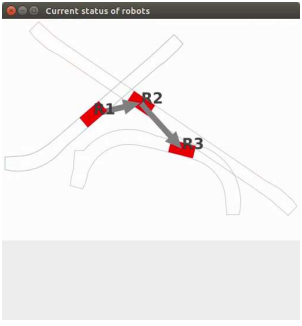
- Precedences (4) used to determine **Critical Points** (5)
- Critical Point (CP): latest path point it is **safe** to navigate to
- CPs are **updated** every T by the coordination algorithm
- If communication is **interrupted***, fleet is left in a safe state



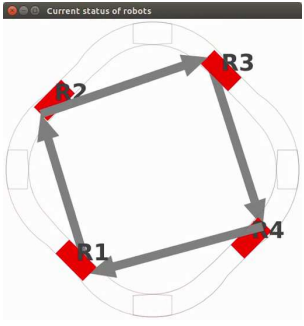
* *communication to all robots ceases*

Toy Examples

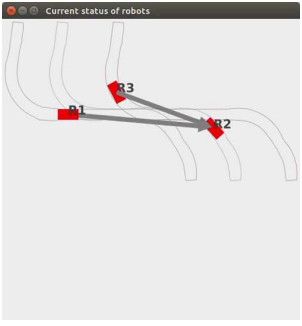
ThreeRobotsLinear



FourRobotsCircle



ThreeRobotsGoalPosting



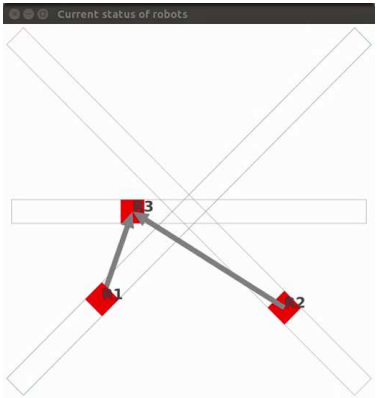
Critical Sections as "Binary" Resources

- Exclusive use of critical section C_{ij} imposed by constraint:

$$\langle \boldsymbol{p}_i, \boldsymbol{p}_j, \inf_{\boldsymbol{p}_i} C_{ij}, \sup_{\boldsymbol{p}_j} C_{ij} \rangle$$

- Guarantees that $\boldsymbol{p}_i(\sigma_i)$ and $\boldsymbol{p}_j(\sigma_j)$ will not collide
- But is inefficient if motion is in the same direction

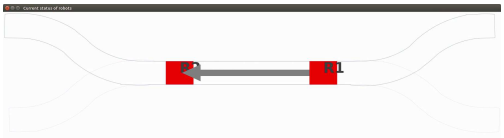
ThreeRobotsSimple



Critical Points and "Following" Behavior

Precedence constraint regulating access to $C_{ij} : \langle \boldsymbol{p}_i, \boldsymbol{p}_j, q_i(t), \sup_{\boldsymbol{p}_j} C_{ij} \rangle$

TwoRobotsFollowing

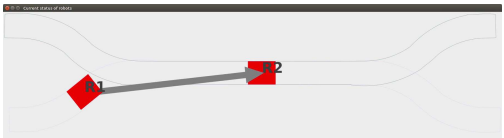


TwoRobotsOpposing

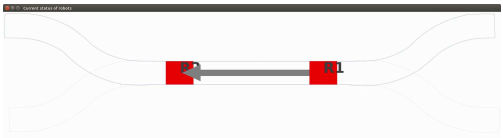
Critical Points and "Following" Behavior

Precedence constraint regulating access to $C_{ij} : \langle \boldsymbol{p}_i, \boldsymbol{p}_j, q_i(t), \sup_{\boldsymbol{p}_j} C_{ij} \rangle$

Paths of robots i and $j : \{\boldsymbol{p}_i, \boldsymbol{p}_j\}$



TwoRobotsFollowing



TwoRobotsOpposing

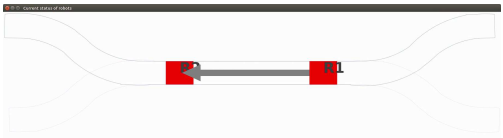
Critical Points and "Following" Behavior

Precedence constraint regulating access to $C_{ij} : \langle \boldsymbol{p}_i, \boldsymbol{p}_j, q_i(t), \sup_{\boldsymbol{p}_j} C_{ij} \rangle$

Paths of robots i and $j : \{\boldsymbol{p}_i, \boldsymbol{p}_j\}$

Portion of path $\boldsymbol{p}$ that has been traversed at time $t : \boldsymbol{p}^{[0,t]}$

TwoRobotsFollowing



TwoRobotsOpposing

Critical Points and "Following" Behavior

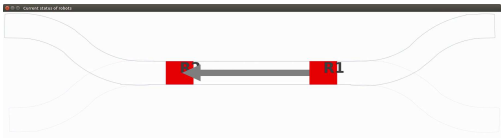
Precedence constraint regulating access to $C_{ij} : \langle \boldsymbol{p}_i, \boldsymbol{p}_j, q_i(t), \sup_{\boldsymbol{p}_j} C_{ij} \rangle$

Paths of robots i and $j : \{\boldsymbol{p}_i, \boldsymbol{p}_j\}$

Portion of path $\boldsymbol{p}$ that has been traversed at time $t : \boldsymbol{p}^{[0,t]}$

Furthest q_i along $\boldsymbol{p}_i$ that does not collide with j at time t : $\text{reach}(\boldsymbol{p}_i, \boldsymbol{p}_j, t)$

TwoRobotsFollowing



TwoRobotsOpposing

Critical Points and "Following" Behavior

Precedence constraint regulating access to $C_{ij} : \langle \boldsymbol{p}_i, \boldsymbol{p}_j, q_i(t), \sup_{\boldsymbol{p}_j} C_{ij} \rangle$

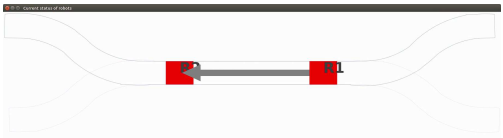
Paths of robots i and $j : \{\boldsymbol{p}_i, \boldsymbol{p}_j\}$

Portion of path $\boldsymbol{p}$ that has been traversed at time $t : \boldsymbol{p}^{[0,t]}$

Furthest q_i along $\boldsymbol{p}_i$ that does not collide with j at time $t : \text{reach}(\boldsymbol{p}_i, \boldsymbol{p}_j, t)$

CP of robot $i : q_i(t) = \begin{cases} \sup_{\boldsymbol{p}_i}\{\inf_{\boldsymbol{p}_i} C_{ij}, \text{reach}(\boldsymbol{p}_i, \boldsymbol{p}_j, t)\}, & \text{if } \sup_{\boldsymbol{p}_j} C_{ij} \notin \boldsymbol{p}_j^{[0,t]} \\ \boldsymbol{p}_i(1), & \text{otherwise} \end{cases}$

`TwoRobotsFollowing`



`TwoRobotsOpposing`

Fleet Coordination as a High-Level Control Loop

Algorithm 2: coordinate(G)

Input: a set G containing goals posted for robots $\{1, \ldots, n\}$.

$\mathcal{P} \leftarrow \emptyset, \mathcal{C} \leftarrow \emptyset, \mathcal{T} \leftarrow \emptyset$

while `true` **do**

- $t \leftarrow$ `getCurrentTime()`
- **for** $i \in [1 \,..\, n]$ **do**
  - $s_i \leftarrow$ `sampleState(`i`)`
- **for** $i : g_i \in G \wedge$ `isIdle(`s_i`)` **do**
  - $G \leftarrow G \setminus \{g_i\}$
  - remove all elements relative to robot i from $\mathcal{P}$ and $\mathcal{C}$
  - $\boldsymbol{p}_i \leftarrow$ `computePath(getConfiguration(`s_i`),`g_i`)`
  - $\mathcal{P} \leftarrow \mathcal{P} \cup \{\boldsymbol{p}_i\}$
- **for** $(\boldsymbol{p}_i, \boldsymbol{p}_{j \neq i}) \in \mathcal{P}^2$ **do**
  - $\mathcal{C} \leftarrow \mathcal{C} \cup$ `getIntersections(`$\mathcal{E}(\boldsymbol{p}_i), \mathcal{E}(\boldsymbol{p}_j)$`)`
- $\mathcal{T} \leftarrow$ `reviseConstraints(`$\mathcal{P}, \mathcal{C}, \mathcal{T}, t, \{s_1, \ldots, s_n\}$`)`
- `updateTrajectories(`$\mathcal{P}, \mathcal{T}$`)`
- **while** `getCurrentTime()` $- t < T$ **do**
  - `sleep(`Δt`)`

Algorithm 3: updateTrajectories($\mathcal{P}$, $\mathcal{T}$)

Input: a set $\mathcal{P}$ of paths, a set $\mathcal{T}$ of temporal constraints.

for $\boldsymbol{p}_i \in \mathcal{P}$ **do**

- $T_i = \{q_i \mid \exists j : \langle \boldsymbol{p}_i, \boldsymbol{p}_j, q_i, q_j \rangle \in \mathcal{T}\}$
- **if** $T_i \neq \emptyset$ **then**
  - $q_i^{\text{closest}} \leftarrow \arg\min_{q_i \in T_i} \boldsymbol{p}_i^{-1}(q_i)$
  - `updateTrajectory(`$\boldsymbol{p}_i, q_i^{\text{closest}}$`)`
- **else**
  - `updateTrajectory(`$\boldsymbol{p}_i, \boldsymbol{p}_i(1)$`)`

Fleet Coordination as a High-Level Control Loop

Algorithm 2: coordinate(G)

Input: a set G containing goals posted for robots $\{1, \ldots, n\}$.

$\mathcal{P} \leftarrow \emptyset, \mathcal{C} \leftarrow \emptyset, \mathcal{T} \leftarrow \emptyset$

while `true` **do**

- $t \leftarrow$ `getCurrentTime()`
- **for** $i \in [1 \,..\, n]$ **do**
  - $s_i \leftarrow$ `sampleState(`i`)`
- **for** $i : g_i \in G \wedge$ `isIdle(`s_i`)` **do**
  - $G \leftarrow G \setminus \{g_i\}$
  - remove all elements relative to robot i from $\mathcal{P}$ and $\mathcal{C}$
  - $\boldsymbol{p}_i \leftarrow$ `computePath(getConfiguration(`s_i`),`g_i`)`
  - $\mathcal{P} \leftarrow \mathcal{P} \cup \{\boldsymbol{p}_i\}$
- **for** $(\boldsymbol{p}_i, \boldsymbol{p}_{j \neq i}) \in \mathcal{P}^2$ **do**
  - $\mathcal{C} \leftarrow \mathcal{C} \cup$ `getIntersections(`$\mathcal{E}(\boldsymbol{p}_i), \mathcal{E}(\boldsymbol{p}_j)$`)`
- $\mathcal{T} \leftarrow$ `reviseConstraints(`$\mathcal{P}, \mathcal{C}, \mathcal{T}, t, \{s_1, \ldots, s_n\}$`)`
- `updateTrajectories(`$\mathcal{P}, \mathcal{T}$`)`
- **while** `getCurrentTime()` $- t < T$ **do**
  - `sleep(`Δt`)`

Algorithm 3: updateTrajectories($\mathcal{P}$, $\mathcal{T}$)

Input: a set $\mathcal{P}$ of paths, a set $\mathcal{T}$ of temporal constraints.

for $\boldsymbol{p}_i \in \mathcal{P}$ **do**

- $T_i = \{q_i \mid \exists j : \langle \boldsymbol{p}_i, \boldsymbol{p}_j, q_i, q_j \rangle \in \mathcal{T}\}$
- **if** $T_i \neq \emptyset$ **then**
  - $q_i^{\text{closest}} \leftarrow \arg\min_{q_i \in T_i} \boldsymbol{p}_i^{-1}(q_i)$
  - `updateTrajectory(`$\boldsymbol{p}_i, q_i^{\text{closest}}$`)`
- **else**
  - `updateTrajectory(`$\boldsymbol{p}_i, \boldsymbol{p}_i(1)$`)`

1 Sample state of all robots

Fleet Coordination as a High-Level Control Loop

Algorithm 2: coordinate(G)

Input: a set G containing goals posted for robots $\{1, \ldots, n\}$.

$\mathcal{P} \leftarrow \emptyset, \mathcal{C} \leftarrow \emptyset, \mathcal{T} \leftarrow \emptyset$

while `true` **do**

- $t \leftarrow$ `getCurrentTime()`
- **for** $i \in [1 \,..\, n]$ **do**
  - $s_i \leftarrow$ `sampleState(`i`)`
- **for** $i : g_i \in G \wedge$ `isIdle(`s_i`)` **do**
  - $G \leftarrow G \setminus \{g_i\}$
  - remove all elements relative to robot i from $\mathcal{P}$ and $\mathcal{C}$
  - $\boldsymbol{p}_i \leftarrow$ `computePath(getConfiguration(`s_i`),`g_i`)`
  - $\mathcal{P} \leftarrow \mathcal{P} \cup \{\boldsymbol{p}_i\}$
- **for** $(\boldsymbol{p}_i, \boldsymbol{p}_{j \neq i}) \in \mathcal{P}^2$ **do**
  - $\mathcal{C} \leftarrow \mathcal{C} \cup$ `getIntersections(`$\mathcal{E}(\boldsymbol{p}_i), \mathcal{E}(\boldsymbol{p}_j)$`)`
- $\mathcal{T} \leftarrow$ `reviseConstraints(`$\mathcal{P}, \mathcal{C}, \mathcal{T}, t, \{s_1, \ldots, s_n\}$`)`
- `updateTrajectories(`$\mathcal{P}, \mathcal{T}$`)`
- **while** `getCurrentTime()` $- t < T$ **do**
  - `sleep(`Δt`)`

Algorithm 3: updateTrajectories($\mathcal{P}$, $\mathcal{T}$)

Input: a set $\mathcal{P}$ of paths, a set $\mathcal{T}$ of temporal constraints.

for $\boldsymbol{p}_i \in \mathcal{P}$ **do**

- $T_i = \{q_i \mid \exists j : \langle \boldsymbol{p}_i, \boldsymbol{p}_j, q_i, q_j \rangle \in \mathcal{T}\}$
- **if** $T_i \neq \emptyset$ **then**
  - $q_i^{\text{closest}} \leftarrow \arg\min_{q_i \in T_i} \boldsymbol{p}_i^{-1}(q_i)$
  - `updateTrajectory(`$\boldsymbol{p}_i, q_i^{\text{closest}}$`)`
- **else**
  - `updateTrajectory(`$\boldsymbol{p}_i, \boldsymbol{p}_i(1)$`)`

1. Sample state of all robots
2. For each idle robot, compute traj. env. to new goal

Fleet Coordination as a High-Level Control Loop

Algorithm 2: coordinate(G)

Input: a set G containing goals posted for robots $\{1, \ldots, n\}$.

$\mathcal{P} \leftarrow \emptyset, \mathcal{C} \leftarrow \emptyset, \mathcal{T} \leftarrow \emptyset$

while `true` **do**

- $t \leftarrow$ `getCurrentTime()`
- **for** $i \in [1 \,..\, n]$ **do**
  - $s_i \leftarrow$ `sampleState(`i`)`
- **for** $i : g_i \in G \wedge$ `isIdle(`s_i`)` **do**
  - $G \leftarrow G \setminus \{g_i\}$
  - remove all elements relative to robot i from $\mathcal{P}$ and $\mathcal{C}$
  - $\boldsymbol{p}_i \leftarrow$ `computePath(getConfiguration(`s_i`),`g_i`)`
  - $\mathcal{P} \leftarrow \mathcal{P} \cup \{\boldsymbol{p}_i\}$
- **for** $(\boldsymbol{p}_i, \boldsymbol{p}_{j \neq i}) \in \mathcal{P}^2$ **do**
  - $\mathcal{C} \leftarrow \mathcal{C} \cup$ `getIntersections(`$\mathcal{E}(\boldsymbol{p}_i), \mathcal{E}(\boldsymbol{p}_j)$`)`
- $\mathcal{T} \leftarrow$ `reviseConstraints(`$\mathcal{P}, \mathcal{C}, \mathcal{T}, t, \{s_1, \ldots, s_n\}$`)`
- `updateTrajectories(`$\mathcal{P}, \mathcal{T}$`)`
- **while** `getCurrentTime()` $- t < T$ **do**
  - `sleep(`Δt`)`

Algorithm 3: updateTrajectories($\mathcal{P}$, $\mathcal{T}$)

Input: a set $\mathcal{P}$ of paths, a set $\mathcal{T}$ of temporal constraints.

for $\boldsymbol{p}_i \in \mathcal{P}$ **do**

- $T_i = \{q_i \mid \exists j : \langle \boldsymbol{p}_i, \boldsymbol{p}_j, q_i, q_j \rangle \in \mathcal{T}\}$
- **if** $T_i \neq \emptyset$ **then**
  - $q_i^{\text{closest}} \leftarrow \arg\min_{q_i \in T_i} \boldsymbol{p}_i^{-1}(q_i)$
  - `updateTrajectory(`$\boldsymbol{p}_i, q_i^{\text{closest}}$`)`
- **else**
  - `updateTrajectory(`$\boldsymbol{p}_i, \boldsymbol{p}_i(1)$`)`

1. Sample state of all robots
2. For each idle robot, compute traj. env. to new goal
3. Update set of critical sections

Fleet Coordination as a High-Level Control Loop

Algorithm 2: coordinate(G)

Input: a set G containing goals posted for robots $\{1, \ldots, n\}$.

$\mathcal{P} \leftarrow \emptyset, \mathcal{C} \leftarrow \emptyset, \mathcal{T} \leftarrow \emptyset$

while `true` **do**

- $t \leftarrow$ `getCurrentTime()`
- **for** $i \in [1 \,..\, n]$ **do**
  - $s_i \leftarrow$ `sampleState(`i`)`
- **for** $i : g_i \in G \wedge$ `isIdle(`s_i`)` **do**
  - $G \leftarrow G \setminus \{g_i\}$
  - remove all elements relative to robot i from $\mathcal{P}$ and $\mathcal{C}$
  - $\boldsymbol{p}_i \leftarrow$ `computePath(getConfiguration(`s_i`),`g_i`)`
  - $\mathcal{P} \leftarrow \mathcal{P} \cup \{\boldsymbol{p}_i\}$
- **for** $(\boldsymbol{p}_i, \boldsymbol{p}_{j \neq i}) \in \mathcal{P}^2$ **do**
  - $\mathcal{C} \leftarrow \mathcal{C} \cup$ `getIntersections(`$\mathcal{E}(\boldsymbol{p}_i), \mathcal{E}(\boldsymbol{p}_j)$`)`
- $\mathcal{T} \leftarrow$ `reviseConstraints(`$\mathcal{P}, \mathcal{C}, \mathcal{T}, t, \{s_1, \ldots, s_n\}$`)`
- `updateTrajectories(`$\mathcal{P}, \mathcal{T}$`)`
- **while** `getCurrentTime()` $- t < T$ **do**
  - `sleep(`Δt`)`

Algorithm 3: updateTrajectories($\mathcal{P}$, $\mathcal{T}$)

Input: a set $\mathcal{P}$ of paths, a set $\mathcal{T}$ of temporal constraints.

for $\boldsymbol{p}_i \in \mathcal{P}$ **do**

- $T_i = \{q_i \mid \exists j : \langle \boldsymbol{p}_i, \boldsymbol{p}_j, q_i, q_j \rangle \in \mathcal{T}\}$
- **if** $T_i \neq \emptyset$ **then**
  - $q_i^{\text{closest}} \leftarrow \arg\min_{q_i \in T_i} \boldsymbol{p}_i^{-1}(q_i)$
  - `updateTrajectory(`$\boldsymbol{p}_i, q_i^{\text{closest}}$`)`
- **else**
  - `updateTrajectory(`$\boldsymbol{p}_i, \boldsymbol{p}_i(1)$`)`

1. Sample state of all robots
2. For each idle robot, compute traj. env. to new goal
3. Update set of critical sections
4. Compute resolving precedence constraints

Fleet Coordination as a High-Level Control Loop

Algorithm 2: coordinate(G)

Input: a set G containing goals posted for robots $\{1, \ldots, n\}$.

$\mathcal{P} \leftarrow \emptyset, \mathcal{C} \leftarrow \emptyset, \mathcal{T} \leftarrow \emptyset$

while `true` **do**

- $t \leftarrow$ `getCurrentTime()`
- **for** $i \in [1 \,..\, n]$ **do**
  - $s_i \leftarrow$ `sampleState(`i`)`
- **for** $i : g_i \in G \wedge$ `isIdle(`s_i`)` **do**
  - $G \leftarrow G \setminus \{g_i\}$
  - remove all elements relative to robot i from $\mathcal{P}$ and $\mathcal{C}$
  - $\boldsymbol{p}_i \leftarrow$ `computePath(getConfiguration(`s_i`),`g_i`)`
  - $\mathcal{P} \leftarrow \mathcal{P} \cup \{\boldsymbol{p}_i\}$
- **for** $(\boldsymbol{p}_i, \boldsymbol{p}_{j \neq i}) \in \mathcal{P}^2$ **do**
  - $\mathcal{C} \leftarrow \mathcal{C} \cup$ `getIntersections(`$\mathcal{E}(\boldsymbol{p}_i), \mathcal{E}(\boldsymbol{p}_j)$`)`
- $\mathcal{T} \leftarrow$ `reviseConstraints(`$\mathcal{P}, \mathcal{C}, \mathcal{T}, t, \{s_1, \ldots, s_n\}$`)`
- `updateTrajectories(`$\mathcal{P}, \mathcal{T}$`)`
- **while** `getCurrentTime()` $- t < T$ **do**
  - `sleep(`Δt`)`

Algorithm 3: updateTrajectories($\mathcal{P}$, $\mathcal{T}$)

Input: a set $\mathcal{P}$ of paths, a set $\mathcal{T}$ of temporal constraints.

for $\boldsymbol{p}_i \in \mathcal{P}$ **do**

- $T_i = \{q_i \mid \exists j : \langle \boldsymbol{p}_i, \boldsymbol{p}_j, q_i, q_j \rangle \in \mathcal{T}\}$
- **if** $T_i \neq \emptyset$ **then**
  - $q_i^{\text{closest}} \leftarrow \arg\min_{q_i \in T_i} \boldsymbol{p}_i^{-1}(q_i)$
  - `updateTrajectory(`$\boldsymbol{p}_i, q_i^{\text{closest}}$`)`
- **else**
  - `updateTrajectory(`$\boldsymbol{p}_i, \boldsymbol{p}_i(1)$`)`

1. Sample state of all robots
2. For each idle robot, compute traj. env. to new goal
3. Update set of critical sections
4. Compute resolving precedence constraints
5. Update critical points of all robots, sleep

Forward Model M and Heuristic h

- Precedences are revised online (`reviseConstraints`)
- Using a forward model M and a heuristic h
- M determines if a precedence is feasible
  e.g., assuming constant acceleration and deceleration with maximum speed (trapeziodal speed profile)
- h decides precedences for pairs of robots through critical sections
  e.g., closest to entrance of critical section goes first

`ThreeRobotsFast`

Forward Model M and Heuristic h

(extrapolation)

Forward model for robot i : $\begin{cases} \ddot{q}_i^{t+\Delta t} & \approx g_i(q_i^t, \dot{q}_i^t, u_i^t), \\ \dot{q}_i^{t+\Delta t} & \approx \dot{q}_i^t + \ddot{q}_i^{t+\Delta t}\Delta t, \\ q_i^{t+\Delta t} & \approx q_i^t + \dot{q}_i^{t+\Delta t}\Delta t, \end{cases}$

Maximum deceleration control for robot i : u_i^{dec}

Current configuration of robot i given its state s_i : $\begin{cases} q_i^0 & = q(s_i) \\ \dot{q}_i^0 & = \dot{q}(s_i) \\ u_i^t & = u_i^{\text{dec}} \end{cases}$

Time at which robot i can stop : $\hat{t}$ such that $\dot{q}_i^{\hat{t}} = 0$

Robot i can yield for robot j at C_{ij} iff $\boldsymbol{p}_i^{-1}(q_i^{\hat{t}}) < \boldsymbol{p}_i^{-1}(\inf_{\boldsymbol{p}_i} C_{ij})$

Set of feasible robot orderings through C_{ij} : $F_{ij} = \{(k, m) \mid \boldsymbol{p}_k^{-1}(q_k^{\hat{t}}) < \boldsymbol{p}_k^{-1}(\inf_{\boldsymbol{p}_k} C_{ij})\}$

Best feasible robot ordering for critical section $C_{ij} = \underset{(k,m)\in F_{ij}}{\arg\min}\, h(s_k, s_m, k, m)$

Formal Properties

Lemma (correctness of constraints)

Given paths $\mathcal{P}$, critical sections $\mathcal{C}$, a complete ordering $\mathcal{T}$ through $\mathcal{C}$, if $\sigma_i(t)$ adhere to $\mathcal{T}$, then the robots will not collide.

Proof (sketch).

Based on definition of constraints and assumption that robot controllers adhere to them. □

Formal Properties: Dynamic Orderings

Theorem (correctness of algorithm)

Algorithm `coordination` *guarantees the absence of collisions if* `reviseConstraints` *decides ordering*

$$(k, m) = \arg\min_{(k,m) \in F_{ij}} h(s_k, s_m, k, m),$$

where F_{ij} are orderings that are "safe" wrt a conservative forward model, and h is any heuristic.

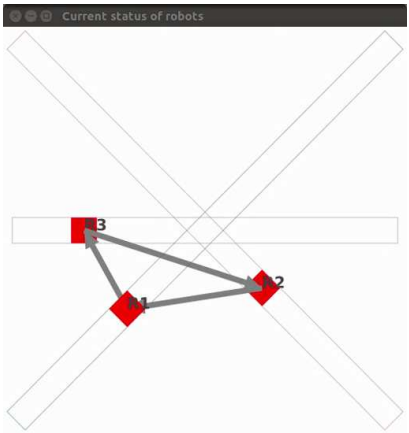
Proof (sketch).

Based on Lemma and the fact that the ordering accounts for dynamic feasibility. □

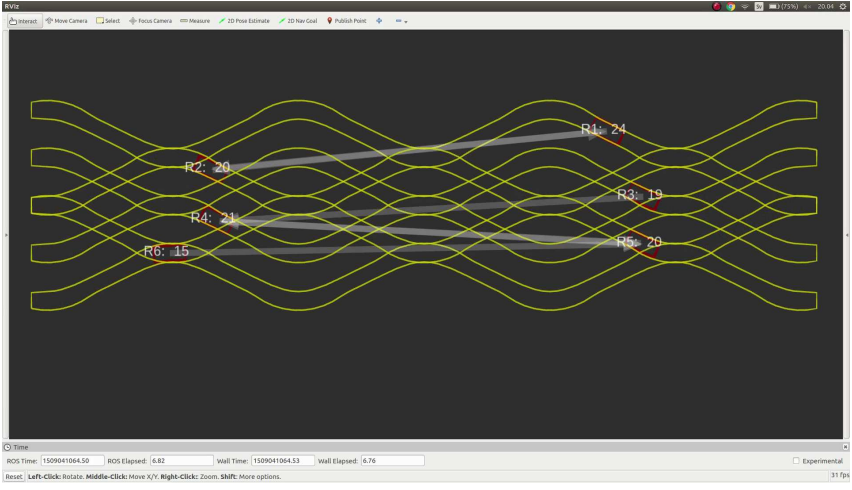
Formal Properties: Deadlocks

- **Cycle** of precedence constraints:
  $\langle \boldsymbol{p}_{i_1}, \boldsymbol{p}_{i_2}, q'_{i_1}, q_{i_2} \rangle, \langle \boldsymbol{p}_{i_2}, \boldsymbol{p}_{i_3}, q'_{i_2}, q_{i_3} \rangle,$
  $\ldots, \langle \boldsymbol{p}_{i_{m-1}}, \boldsymbol{p}_{i_m = i_1}, q'_{i_{m-1}}, q_{i_m = i_1} \rangle$
- Cycle is **unsafe** iff:
  $\boldsymbol{p}_{i_j}^{-1}(q_{i_j}) > \boldsymbol{p}_{i_j}^{-1}(q'_{i_j})$ for all $j \in [2\,..\,m]$
- Unsafe cycles lead to **deadlocks**
- Trajectories are **deadlock-free** if
  - no unsafe cycles, and
  - robots do not start/end in critical sections

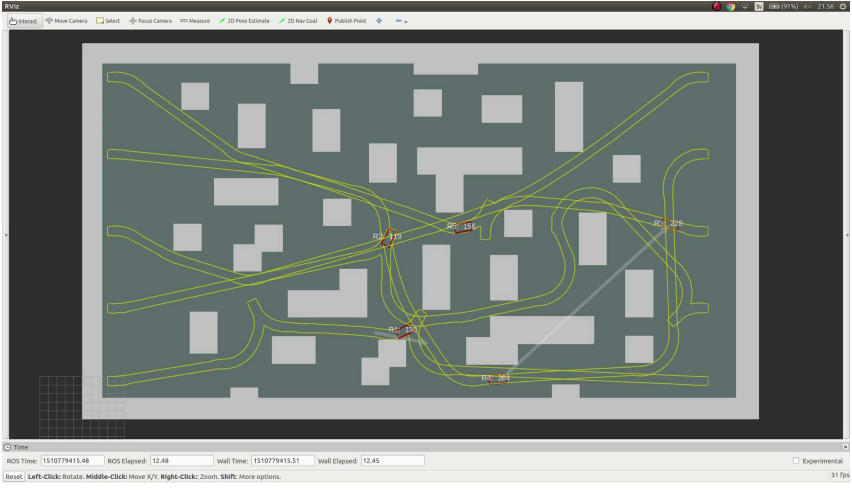
ThreeRobotsDeadlock



Six Robots Traversing Fixed Paths



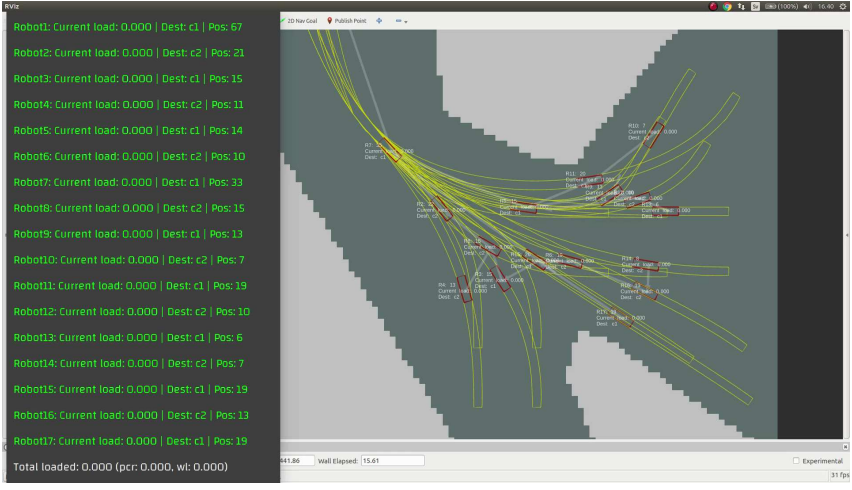
Five Robots Traversing a Random Environment



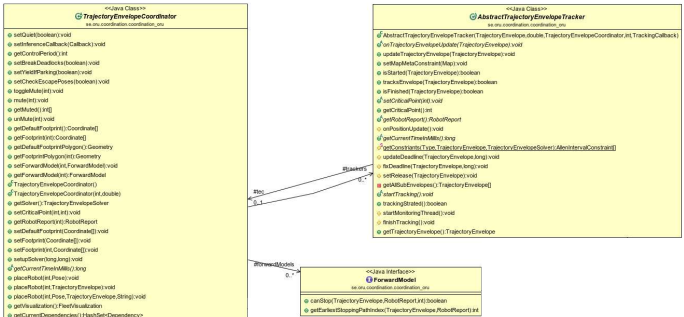
Two Forklifts in AASS Basement (Reliability Study)

Reacting to Contingencies with Real Robots

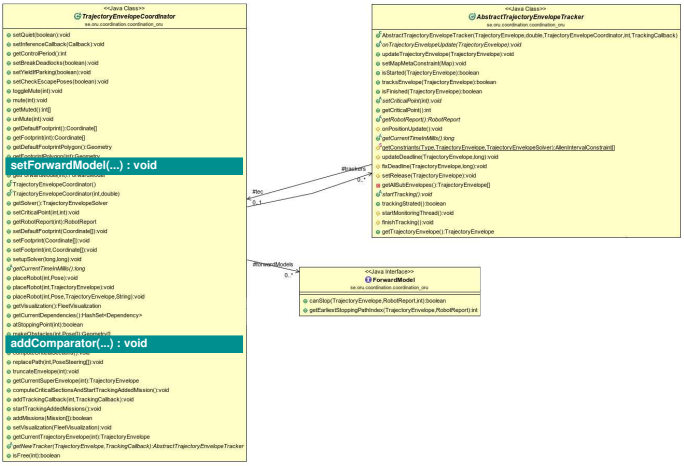
Industrial Use-Case: 17 Robots in a Quarry



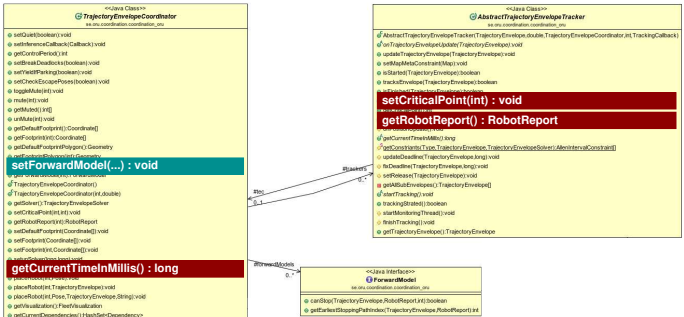
The `coordination_oru` Library



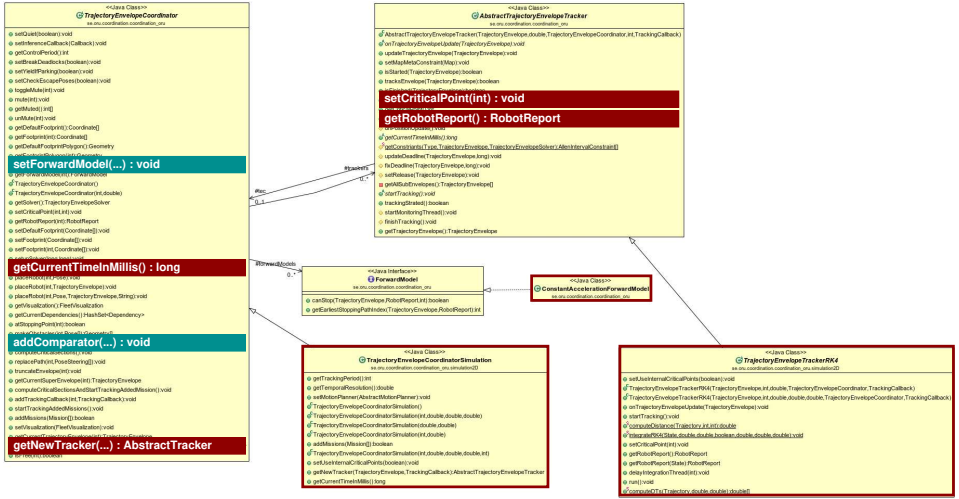
The `coordination_oru` Library: Heuristic h and Forward Model M



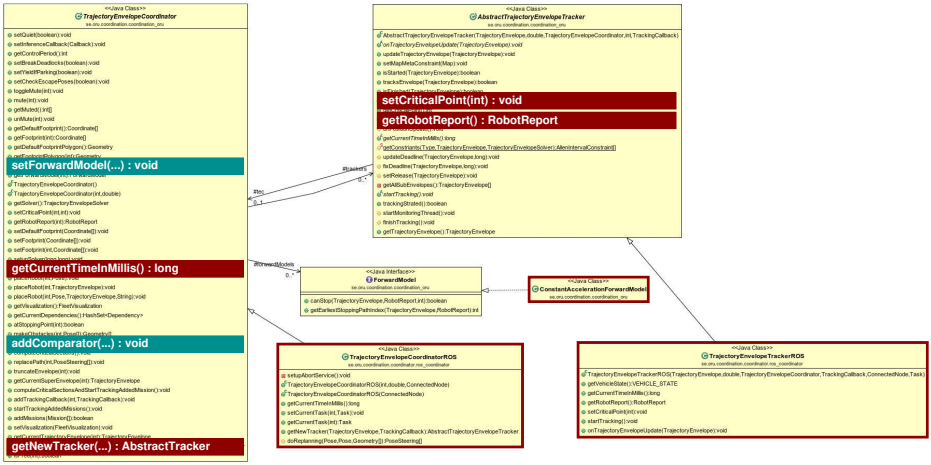
The `coordination_oru` Library: Implementing Interfaces to Robot System



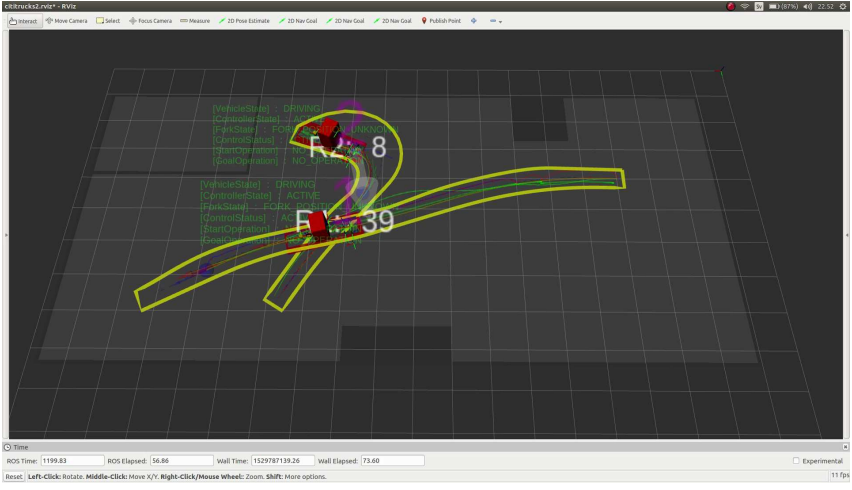
The `coordination_oru` Library: Interface to Built-In 2D Simulator



The `coordination_oru` Library: Interface to ROS



Point and Click Goal Posting in ROS/Gazebo



Outline

Summary

- **Fleet automation** subsumes many tightly connected problems
- We have seen three examples which make use of **trajectory envelopes**
- In all three, coordination is a realized as a **high-level control loop**
- We have seen two forms of coordination, with complementary features
  - Systematic: guarantees absence of collisions, deadlocks, adherence to temporal constraints
  - Online: optimize time to completion, guarantee absence of collisions, scale well
- **Key points** in all three examples
  - exploit spatio-temporal representation
  - compute updated profiles online
  - communicate with robot controllers when necessary

Thank you!

semanticrobots.oru.se

iliad-project.eu

References I

Andreasson, H., Bouguerra, A., Cirillo, M., Dimitrov, D., Driankov, D., Karlsson, L., Lilienthal, A., Pecora, F., Saarinen, J., Sherikov, A., and Stoyanov, T. (2015).
Autonomous transport vehicles: where we are and what is missing.
IEEE Robotics & Automation Magazine, 22(1):64–75.

Cirillo, M., Pecora, F., Andreasson, H., Uras, T., and Koenig, S. (2014).
Integrated Motion Planning and Coordination for Industrial Vehicles.
In *Proc. of the 24th Int'l Conf. on Automated Planning and Scheduling (ICAPS)*.

Culberson, J. C. (1997).
Sokoban is PSPACE-complete.
Technical Report TR 97-02, Department of Computing Science, The University of Alberta, Edmonton, Alberta, Canada.

Liang, T.-C., Liu, J.-S., Hung, G.-T., and Chang, Y.-Z. (2005).
Practical and flexible path planning for car-like mobile robot using maximal-curvature cubic spiral.
Robotics and Autonomous Systems, 52(4):312 – 335.

Mansouri, M., Andreasson, H., and Pecora, F. (2015).
Towards hybrid reasoning for automated industrial fleet management.
In *Workshop on Hybrid Reasoning at IJCAI'15, Buenos Aires, Argentina, 25-31 July, 2015.*

Mansouri, M., Andreasson, H., and Pecora, F. (2016).
Hybrid reasoning for multi-robot drill planning in open-pit mines.
Acta Polytechnica, 56(1):47–56.

References II


Mansouri, M., Lagriffoul, F., and Pecora, F. (2017).
Multi vehicle routing problem with nonholonomic constraints and dense dynamic obstacles.
In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE.


Pecora, F., Andreasson, H., Mansouri, M., and Petkov, V. (2018).
A loosely-coupled approach for multi-robot coordination, motion planning and control.
In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*.


Pecora, F., Cirillo, M., and Dimitrov, D. (2012).
On mission-dependent coordination of multiple vehicles under spatial and temporal constraints.
In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.


Toth, P. and Vigo, D., editors (2001).
The Vehicle Routing Problem.
Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.


Váňa, P. and Faigl, J. (2015).
On the dubins traveling salesman problem with neighborhoods.
In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4029–4034.