Distributed Aerial Robotics

Aerial Swarm Tools and Applications
RSS 2024 Workshop — July 19, Delft (NL)

Lorenzo Pichierri | Andrea Testa | Giuseppe Notarstefano

Department of Electrical, Electronic, and Information Engineering
Alma Mater Studiorum Università di Bologna

lorenzo.pichierri@unibo.com, a.testa@unibo.com, giuseppe.notarstefano@unibo.it

RESEARCH FUNDED BY
Project BR22GR01
"Distributed Optimization for Cooperative Machine Learning in Complex Networks”

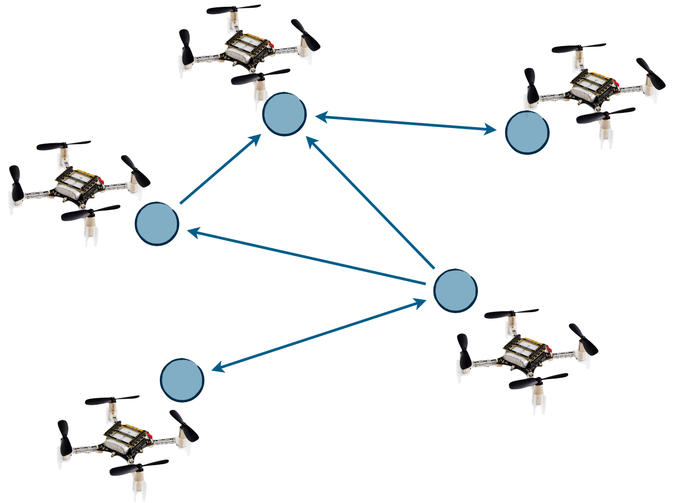
Ministero degli Affari Esteri
e della Cooperazione Internazionale

Distributed Multi-Robot research

Networks of (heterogeneous) communicating robots

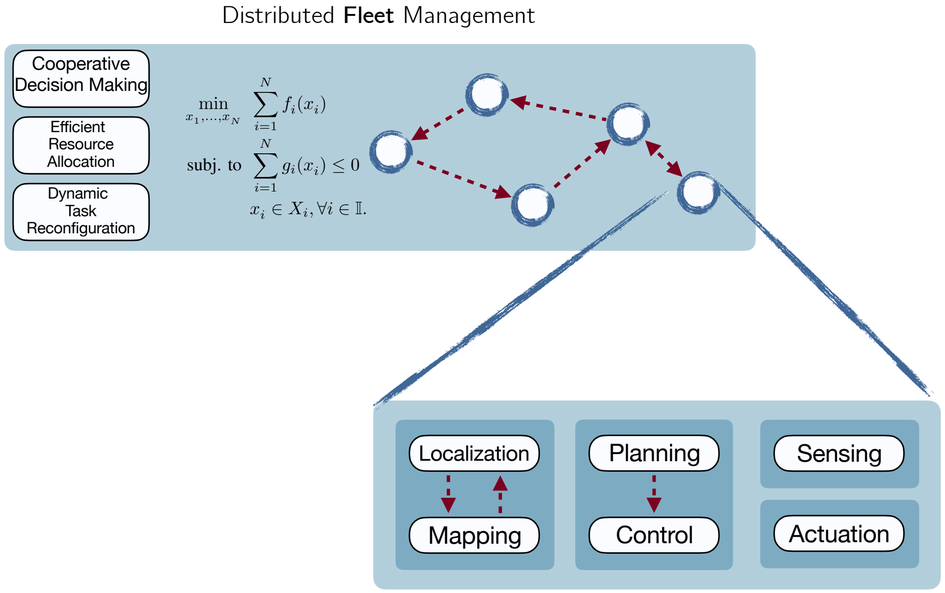
Each robot has computation and communication capabilities

Goal: solve complex problem leveraging *sparse* communication

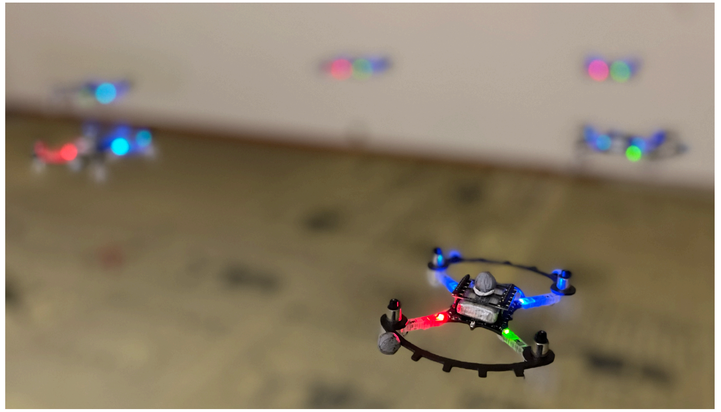


Relevant Applications

- Pick-up and Delivery Vehicle Routing
- Task Allocation
- Surveillance and Monitoring
- Patrolling and Encirclement



Toolbox for Distributed Aerial Robotics



Scenario

Swarm of Crazyflies for complex cooperative tasks in a *decentralized* fashion

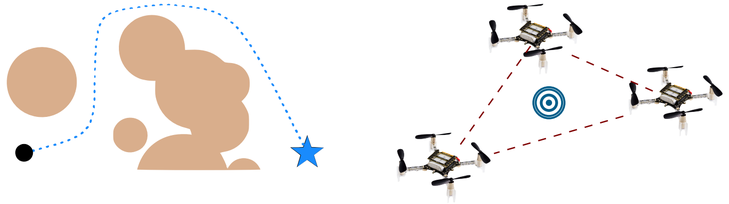
- ROS 2 Middelware
- Bitcraze Firmware Bindings
- Webots Physical Engine

Main Features

- Online distributed control and optimization
- Full-stack swarm management
- Mixed-Real experiments via realistic CAD-based simulations

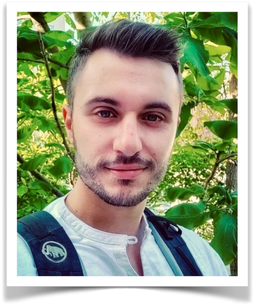
News 🎉

- Docker 🐋 — Install CrazyChoir via Docker (readme)
- Compatibility with Loco Positioning System (UWB) and Lighthouse System
- Decentralize Collision Avoidance based on Safety Filters
- Aggregative Optimization Framework for surveillance and monitoring missions



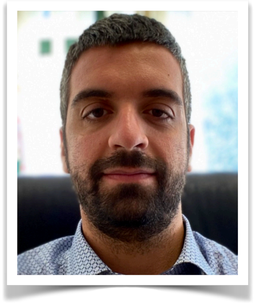
Thanks to…

Dr. Andrea Testa

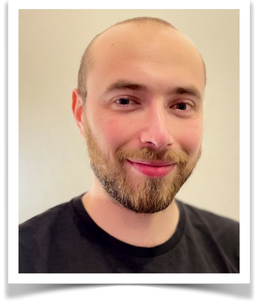


Dr. Lorenzo Pichierri

Dr. Andrea Camisa



Prof. Ivano Notarnicola



Dr. Lorenzo Sforni

Dr. Guido Carnevale

Dr. Alice Rosetti

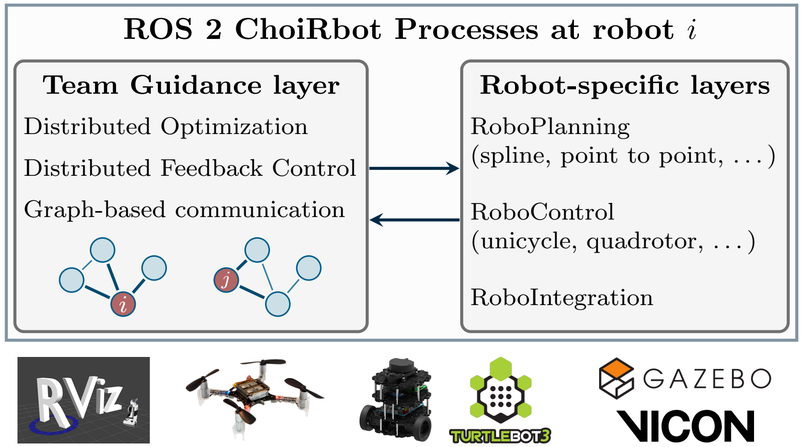
And also…

Marco Borghesi, Simone Baroncini, Andrea Tramaloni, Matteo Sartoni and Riccardo Brumali

ChoiRbot Toolbox for Swarm Robotics

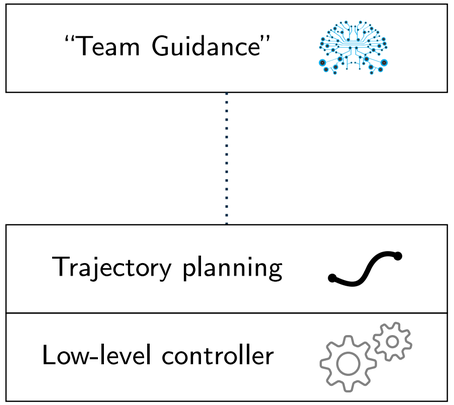
ChoiRbot 2

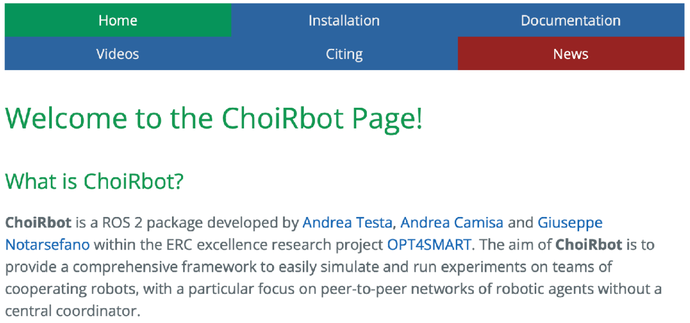
- ROS 2 based toolbox for cooperative robotics
- Robot-to-robot communication
- Distributed intelligence (NO central computation unit)



End-to-end fleet control

- Distributed algorithms for swarm control
  (e.g., pickup & delivery, surveillance)
- From Team Guidance to low-level control
- Heterogeneous robot simulation and experiments





https://opt4smart.github.io/ChoiRbot

Testa et al., "ChoiRbot: A ROS 2 toolbox for cooperative robotics", in 2021 IEEE Robotics and Automation Letters

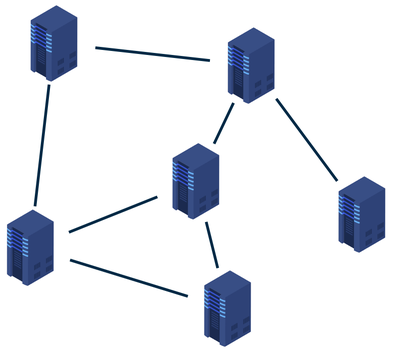
DISROPT Toolbox for Distributed Optimization & AI

DISROPT

- Toolbox for distributed optimization
- Supports Deep Learning libraries (TensorFlow, PyTorch)
- Semantic modeling of optimization problems
- Parallel programming (MPI)

Paradigm

- Peer agents without a master (distributed network computing)
- Very-large scale optimization (big-data analytics)
- Inter-processors communication (unstructured, asynchronous)
- Distributed optimization solvers (nonconvex, MILP)



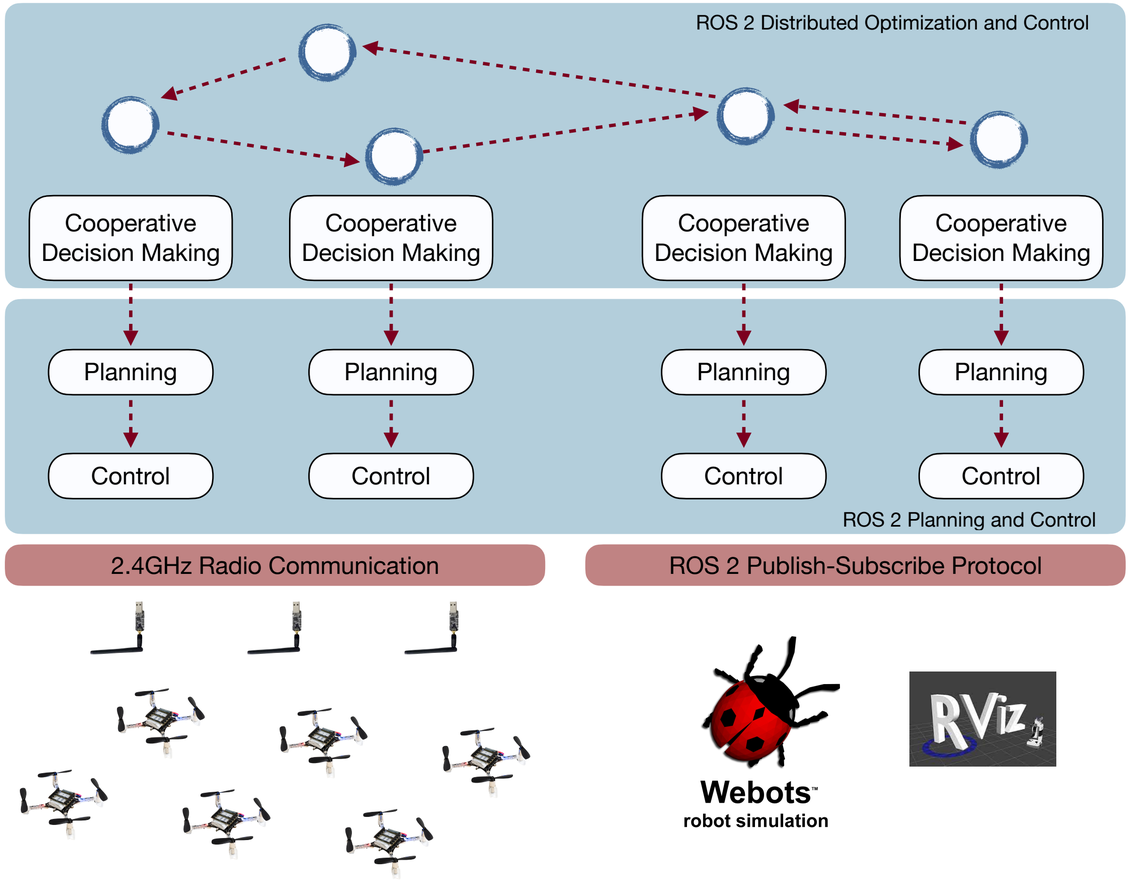
https://opt4smart.github.io/disropt

Farina et al., "Disropt: a python framework for distributed optimization", in 2020 IFAC-PapersOnLine

CrazyChoir — Flying Swarms of Crazyflie Quadrotors in ROS 2

CrazyChoir Layers

- **Cooperative decision-making**
  provides methods to implement *distributed*, *online* optimization and control schemes
- **Planning**
  computes desired trajectory references
- **Control**
  provides classes to implement feedback-control schemes
- **Radio communication**
  leverage ROS 2 communication protocols to provide ad-hoc functions to transmit data to Crazyflies
- **Simulation**
  bridges reality in *Webots* simulator and provides light-weight simulations integrating physics visualized in *RViz*



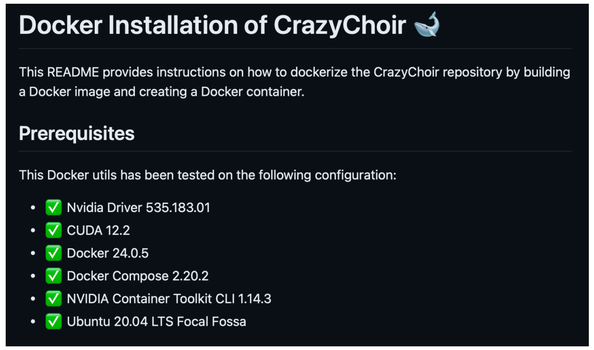
CrazyChoir GUI functionalities

- Hand-draw trajectories, tuning time and interpolation distance
- Manage classical operations (e.g., start, land, takeoff, go_to)

https://opt4smart.github.io/crazychoir

Pichierri et al., "CrazyChoir: Flying Swarms of Crazyflie Quadrotors in ROS 2", in 2023 IEEE Robotics and Automation Letters

How to install…



1. Check Docker and Nvidia prerequisites
2. Clone the repo

```
mkdir -p ~/crazychoir_ws/src
cd ~/crazychoir_ws/src
git clone --recursive https://github.com/OPT4SMART/crazychoir.git
git submodule update --remote --merge ChoiRbot/
```

3. Build the Docker image

```
bash crazychoir_ws/src/docker/buildImage crazychoir_ws/src/docker <image_name>:<tag>
```

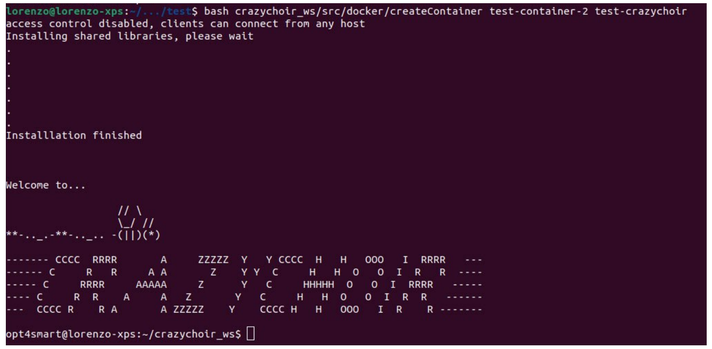
4. Create the container

```
bash crazychoir_ws/src/docker/createContainer <container_name> <image_name>:<tag>
```

5. (Next times) Start the container and execute it

```
docker start <container_name>
docker exec -it <container_name> /bin/bash
```

… *Have Fun!* 🚀



Hands on with… Aggregative Optimization

Aggregative Tracking Algorithm

Aggregative Tracking Optimization Strategy

Initialize the algorithm

- A surveillance team of N robots cooperatively protects a *moving target* from a team of N intruders.
- $x_i^0 = x_i^{init}, s_i^0 = x_i^{init}, y_i^0 = \nabla_1 f_i(x_i^0, s_i^0)$
- Defenders communicate *locally* according to undirected graph.

For $t = 0, 1, \ldots$ agents concurrently perform

- Surveillance scenario modelled by *aggregative optimization*
- Solution update

$$\min_{x \in \mathbb{R}^{2N}} \sum_{i=1}^{N} f_i^t\left(x_i, \sigma(x)\right)$$

$$z_i^{t+1} = z_i^t - \alpha\left(\nabla_1 f(x_i^t, s_i^t) + \nabla\phi(z_i^t) \cdot y_i^t\right)$$

- $x_i \in \mathbb{R}^2$ position of robots
- Trackers update
- $\sigma(x) \triangleq \frac{1}{N}\sum_{i=1}^{N} \phi(x_i)$ aggregative variable

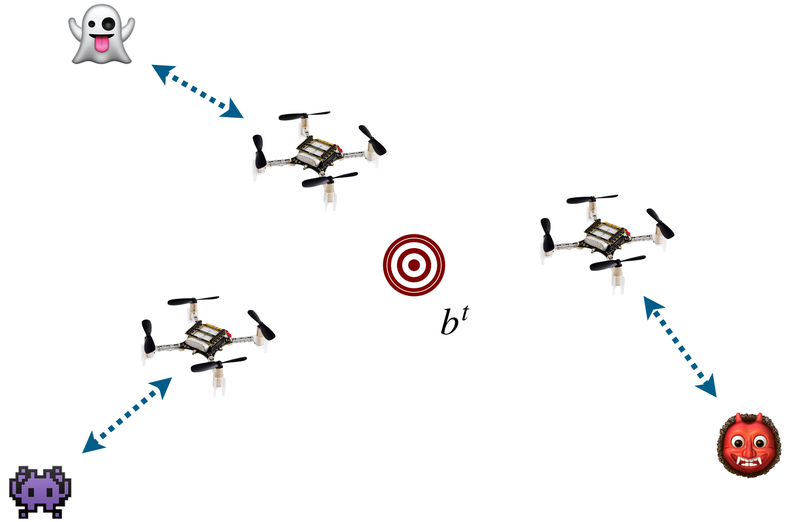
$$s_i^{t+1} = \sum_{j \in \mathcal{N}_i^t} a_{ij}^t s_j^t + \phi_i(x_i^{t+1}) - \phi_i(x_i^t)$$

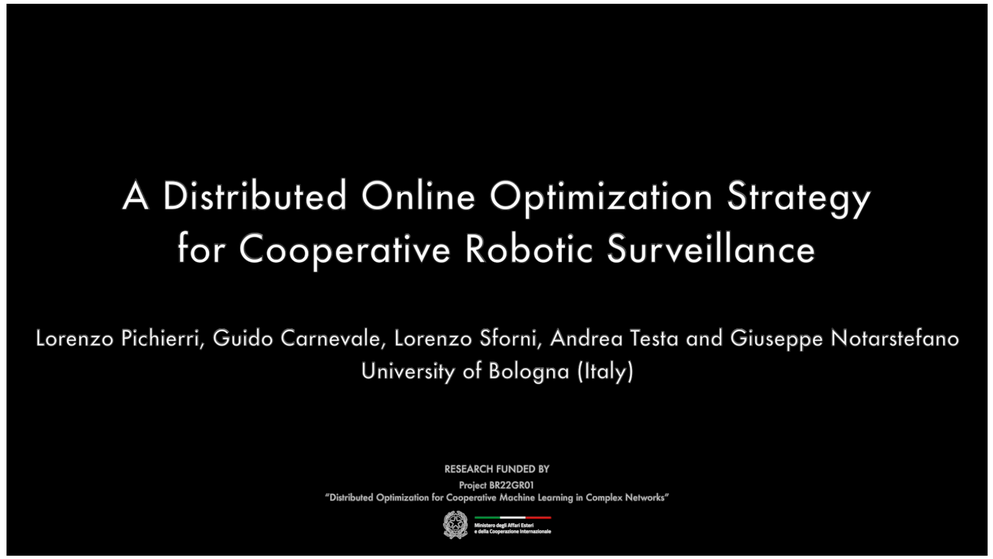
- $f_i^t : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ *local* cost function defined as:

$$y_i^{t+1} = \sum_{j \in \mathcal{N}_i^t} a_{ij}^t y_j^t + \nabla_2 f_i^{t+1}(x_i^{t+1}, s_i^{t+1}) - \nabla_2 f_i^t(x_i^t, s_i^t)$$

$$f_i^t(\,\cdot\,) = \gamma_1 \|x_i - r_i^t\|^2 + \gamma_2 \|\sigma - b^t\|^2 + \gamma_3 \|x_i - \sigma\|^2$$

- $b^t \in \mathbb{R}^2$ target, $r_i^t \in \mathbb{R}^2$ intruders, $\gamma_i > 0$ tradeoff parameters.



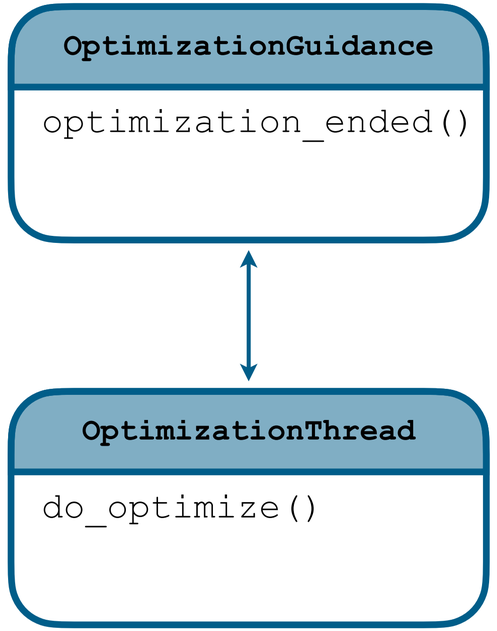


How to implement an algorithm in CrazyChoir

`OptimizationGuidance()`

Leverage ChoiRbot class for optimization-based scenarios

- Provides communication with neighboring robots
- Abstract method `optimization_ended` (to be implemented) called at end of optimization
- Optimization performed in a separate thread to allow for ROS callbacks in main thread



`OptimizationThread()`

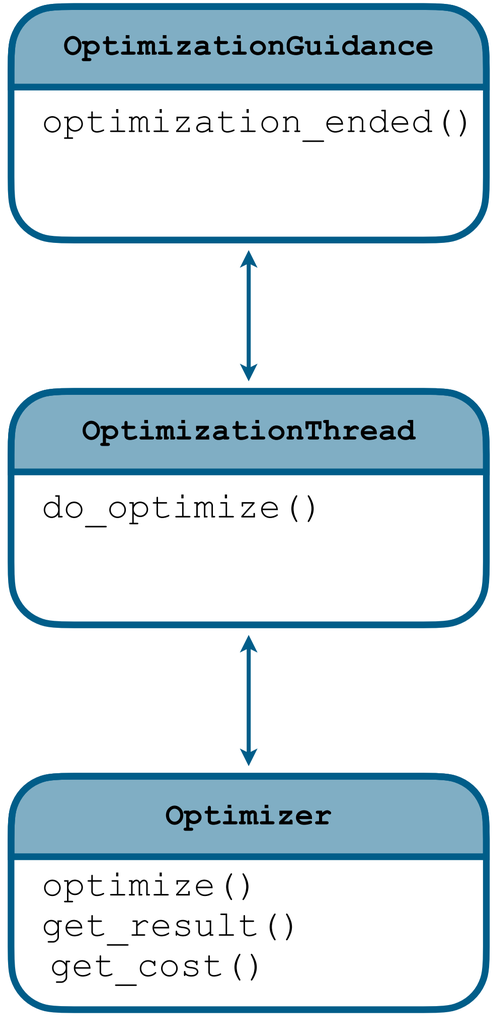
- Provides method for
  - Starting optimization
  - Halting optimization
  - Quitting thread
- ChoiRbot takes care of proper communication between threads
- Abstract method `do_optimize` (to be implemented) to perform optimization

How to implement an algorithm in CrazyChoir (cont'd)

`Optimizer()`

Domain-specific class to call appropriate numerical optimization algorithms

- Optimization problem set-up (variables, cost function, constraints, …)
- Optimization algorithm creation and destruction
- Post-processing of optimization result
- Abstract methods to be implemented
  - **`optimize()`** - actual optimization routine
  - **`get_result()`** - returns properly formatted optimization result
  - **`get_cost()`** - returns optimal cost



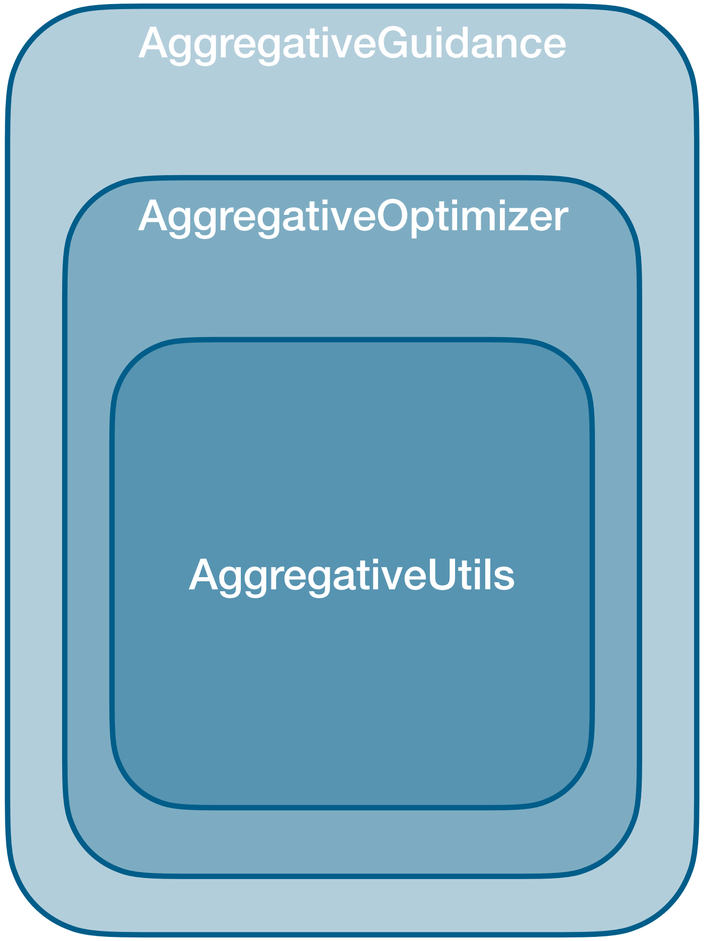
How to implement the Aggregative Tracking

`AggregativeCrazyflieGuidance`

- Import `AggregativeGuidance` class from ChoiRbot
- Initialize methods and settings
- Specify tailored methods for Crazyflies
  - `get_initial_condition()`
  - `get_intruder()`
  - `get_target()`
  - `generate_reference_msg(msg_type, x_des)`

`AggregativeOptimizer`

- Runs the `OptimizationThread`
- Problem set-up and algorithm initialization
  - `initialize(guidance, halt_event,)`
  - `create_problem()`
  - `generate_cost(intruder, target, gammas)`



How to implement the Aggregative Tracking (cont'd)

`AggregativeUtils`

- Defines the Aggregative Problem using DISROPT
- Implements the Aggregative Tracking algorithm
  - `_update_local_solution(x)`
  - `iterate_run(stepsize)`
- Provide post-analysis methods
  - `get_result()`
  - `get_cost()`
  - `get_subgradient()`
  - `get_trackers()`

```python
def iterate_run(self, stepsize: float, **kwargs):
    """Run a single iterate of the aggregative gradient tracking algorithm
    """

    d=self.x.shape[0]
    mask_x = np.hstack((np.eye(d),np.zeros((2,2)))).T
    mask_sigma = np.hstack((np.zeros((2,2)),np.eye(d))).T

    data_received = self.agent.neighbors_exchange([self.s,self.y])
    for neigh in data_received:
        self.s_neigh[neigh] = deepcopy(data_received[neigh][0])
        self.y_neigh[neigh] = deepcopy(data_received[neigh][1])

    s_kp = self.agent.in_weights[self.agent.id] * self.s
    y_kp = self.agent.in_weights[self.agent.id] * self.y

    for j in self.agent.in_neighbors:
        s_kp += self.agent.in_weights[j] * self.s_neigh[j]
        y_kp += self.agent.in_weights[j] * self.y_neigh[j]

    self.nabla_1 = mask_x.T@self.agent.problem.objective_function.subgradient(np.vstack((self.x,self.s)))
    self.nabla_phi = self.agent.problem.aggregative_function.jacobian(self.x).T

    x_kp = self.x - stepsize*(
            mask_x.T@self.agent.problem.objective_function.subgradient(np.vstack((self.x,self.s)))
                + self.agent.problem.aggregative_function.jacobian(self.x).T@self.y)
    s_kp += self.agent.problem.aggregative_function.eval(x_kp) - self.agent.problem.aggregative_function.eval(self.x)

    old_nabla_2 = self.nabla_2
    self.nabla_2 = mask_sigma.T@self.agent.problem.objective_function.subgradient(np.vstack((x_kp,s_kp)))

    y_kp += self.nabla_2 - old_nabla_2

    self._update_local_solution(x_kp, **kwargs)
    self._update_local_aggregative_tracker(s_kp, **kwargs)
    self._update_local_gradient_tracker(y_kp, **kwargs)
```

How to run a Simulation — Organization

Main Packages

These packages gather all the tools needed for cooperative simulation and experiments

- **`ChoiRbot`**
- **`crazychoir`**

Example Package

This package provides all the example to implement novel cooperative, distributed applications

- **`crazychoir_examples`**

`crazychoir_examples`

This folder gather all the executable node that are included in the launch files

`launch`

Here, all the `launch.py` files are collected

`resource`

The Crazyflie driver for Webots setup are included here

`worlds`

It contains all the `.wbt` files for the generation of new world in Webots

How to run a Simulation — Launch file

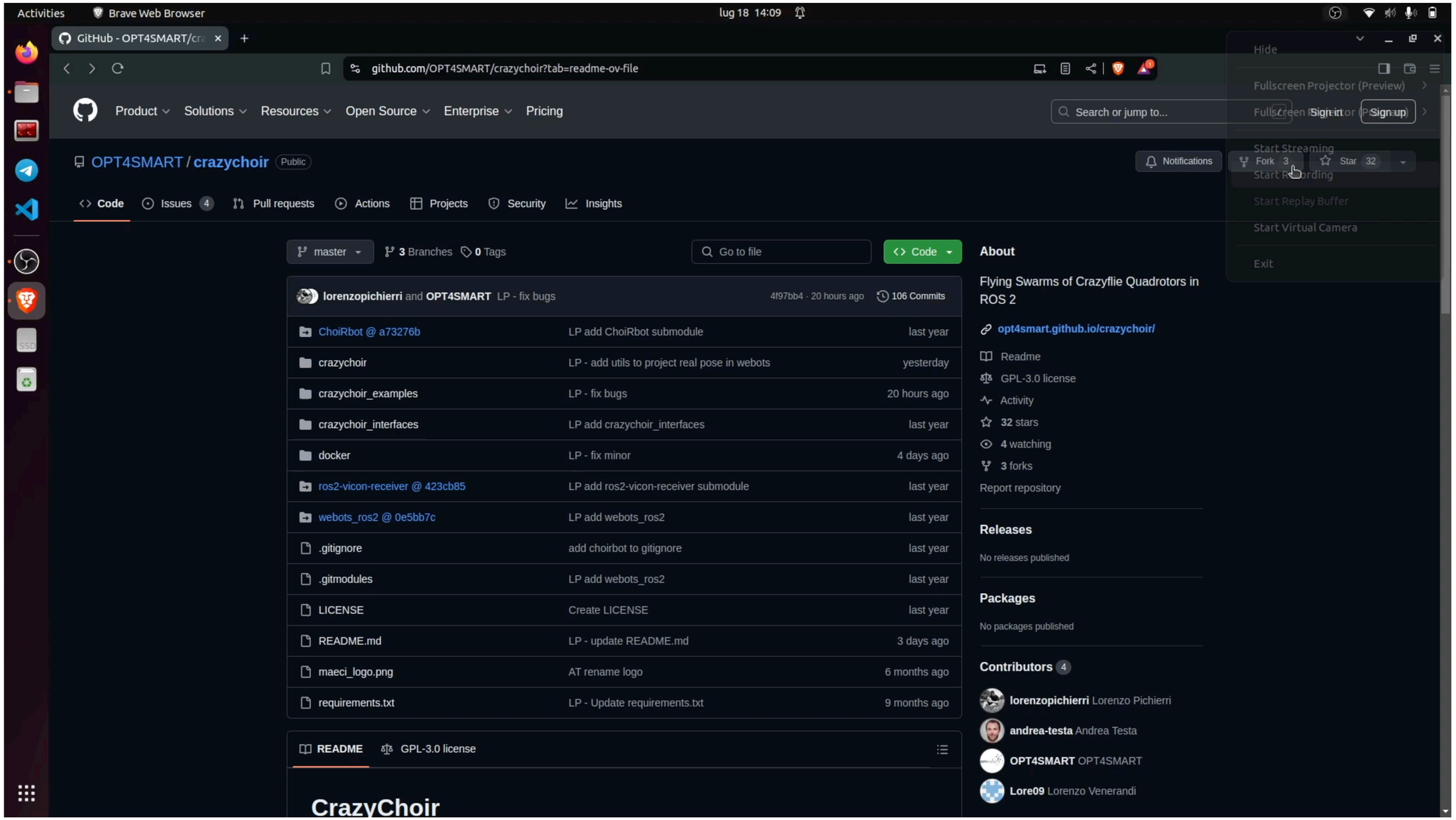
1. Setup Simulation settings
   - Number of Agents, initial positions
   - Nodes frequency positions
   - Communication Graph

2. Define main CrazyChoir layers
   - Guidance Agents: `AggregativeCrazyflieGuidance()`
   - Guidance Intruders: `GoToGuidance()`
   - Optimizer: `AggregativeOptimizer()`
   - Controller: `VelocityController()`
   - Planner: `SplinePosition()`

3. Define Webots methods
   - Generate World (agents, intruders, target)
   - Include Webots driver for Crazyflies

```python
def generate_launch_description():

    N = 3 # number of agents

    Adj = binomial_random_graph(N, p=0.5, seed=3)

    # Initial Position
    P = np.random.uniform(-3, 3, (2*N, 3))

    # Target
    target = [0.0, 0.0, 1.0]

    # Generate Webots World
    generate_webots_world_file(robots, source_filename, target_filename)
    webots = WebotsLauncher(world=target_filename)
    launch_description.append(webots)

    # add executables for each robot
    for i in range(2*N):

        if i < N:
            # Guidance Agents
            launch_description.append(Node(
                package='crazychoir_rss',
                executable='crazychoir_aggregative_webots_guidance_aggregative_moving'))

        else:
            # Guidance Intruder
            launch_description.append(Node(
                package='crazychoir_rss',
                executable='crazychoir_aggregative_webots_guidance_intruder'))

        # Simple guidance (takeoff, landing, stop)
        launch_description.append(Node(
            package='crazychoir_rss',
            executable='crazychoir_aggregative_webots_simple_guidance'))

        # Controller
        launch_description.append(Node(
            package='crazychoir_rss',
            executable='crazychoir_aggregative_webots_controller'))

        # Planner
        launch_description.append(Node(
            package='crazychoir_rss',
            executable='crazychoir_aggregative_webots_trajectory'))

        # Webots Driver
        launch_description.append(get_webots_driver_cf(i))
        launch_description.append(Node(
            package='robot_state_publisher',
            executable='robot_state_publisher'))

    return LaunchDescription(launch_description)
```

How to run a Simulation — Video Tutorial



How to run an Experiment — What to change?

Communication Interface

To communicate with the Crazyflies, you should include the radio node.

`RadioHandler()`

- This is the main class and the abstract methods to be implement are
  - `cmd_sender(msg, uri, cf)`
  - `cmd_stop()`
  - `cmd_take_off()`
  - `set_values()`

- Examples of subclasses are:
  - `RadioHandlerXYZ()` — sends position setpoints and (possibly) groundtruth information
  - `RadioHandlerFPQR()` — sends thrust and angular rates setpoints
  - `RadioHandlerVel()` — sends linear velocity setpoints

How to run an Experiment — What to change? (Cont'd)

1. Modify the pose handler in the executable file
   - If you use the Vicon MoCap, you should include also a pose callback to compute linear and angular velocity
   - With LocoPositioning (UWB) and Lighthouse, you need to modify the topic handler in `'odom_cf'`

2. Include the uris list and the radio node in the launcher file and (if needed) remove the webots tools.

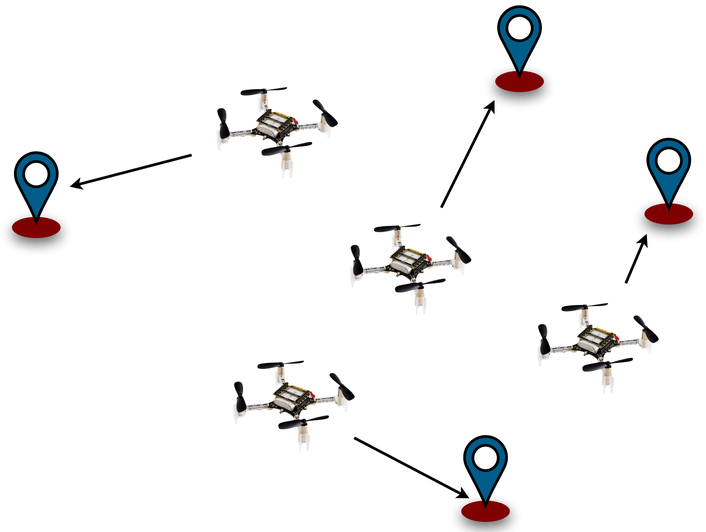
```python
def main():
    rclpy.init()

    frequency = 100

    position_ctrl = FlatnessPositionCtrl(update_frequency= frequency, vicon= True)
    desired_acceleration = FullStateTraj()
    attitude_ctrl = GeometryAttitudeCtrl(update_frequency=frequency, vicon=True)
    sender = FPQRSender(frequency)

    callback = Estimator()
    controller = HierarchicalController(
                pose_handler='vicon', pose_topic='/vicon/cf1/cf1', pose_callback=callback,
                position_strategy=position_ctrl, attitude_strategy=attitude_ctrl,
                command_sender=sender, traj_handler=desired_acceleration)

    callback.set_pose(controller.current_pose)

    controller.get_logger().info('Go!')

    rclpy.spin(controller)
    rclpy.shutdown()
```

```python
# Set uri address for each quadrotor
uris = ['radio://0/80/2M/E7E7E7E702', 'radio://0/90/2M/E7E7E7E704',
        'radio://0/100/2M/E7E7E7E706', 'radio://0/100/2M/E7E7E7E709' ]
# Launch radio node
for r in radios:
    radio_launch.append(Node(
        package='crazychoir_rss',
        executable='crazychoir_aggregative_lighthouse_radio',
        namespace=f'radio_{r}',
        parameters=[{'uris': uris_r,'agent_ids': agent_ids_r,
                     'cmd_topic': 'cmd_vel','logger' : True, }]))
```

```python
def main():
    rclpy.init()

    frequency = 10

    velocity_strategy = VelocityCtrlStrategy(frequency)
    sender = VelSender(frequency)
    desired_reference = PositionTraj()

    controller = VelocityController(pose_handler='pubsub', pose_topic='cf_odom', pose_callback=None,
                velocity_strategy=velocity_strategy,
                command_sender=sender, traj_handler=desired_reference)

    controller.get_logger().info('Go!')

    rclpy.spin(controller)
    rclpy.shutdown()
```

Experiment 

Hands on with… Task Assignment



Problem Setup

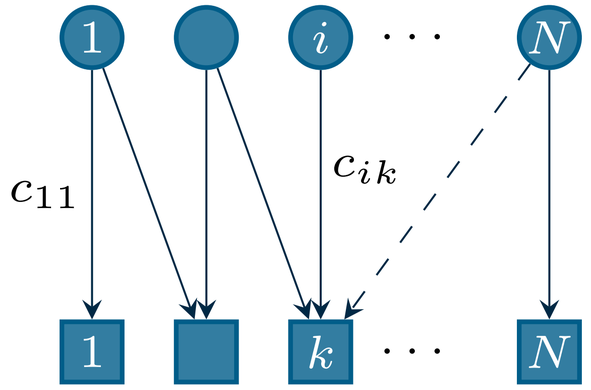
- N robots have to perform N tasks
- $x_{i\kappa} = 1$ if agent i is assigned to task κ, otherwise $x_{i\kappa} = 0$
- $c_{i\kappa}$ cost if agent i performs task κ (e.g., euclidean distance from robot i to position

$$\min \sum_{i=1}^{N} \sum_{\kappa=1}^{N} c_{i\kappa} x_{i\kappa}$$

$$\text{subj . to} \sum_{i=1}^{N} x_{i\kappa} = 1, \qquad k = 1,\dots,N$$

$$\sum_{\kappa=1}^{N} x_{i\kappa} = 1, \qquad i = 1,\dots,N$$

$$0 \leq x_{i\kappa} \leq 1, \qquad i,\kappa = 1,\dots,N$$



Goal

Find a *one-to-one* assignment of robot and tasks that minimizes overall cost

How to run a Simulation — Launch file

1. Setup Simulation settings
   - Number of Agents, initial positions
   - Nodes frequency positions
   - Communication Graph

2. Define main CrazyChoir layers
   - Guidance Agents: `TaskGuidance()`
   - Optimizer: `TaskOptimizer()`
   - Table: `PositionTaskTable()`
   - Planner: `PointToPointPlanner_2D()`

3. Define Webots methods
   - Generate World (agents, task locations)
   - Include Webots driver for Crazyflies

```python
def generate_launch_description():

    # number of agents
    N = 6

    # generate communication graph (this function also sets the seed)
    Adj = binomial_random_graph(N, 0.2, seed=3)

    # generate initial positions in [-3, 3] with z = 0
    P = np.random.uniform(-3, 3, (N, 3))

    # initialize launch description
    launch_description = [] # launched immediately

    # Task table
    launch_description.append(Node(
            package='crazychoir_examples',
            executable='crazychoir_task_assignment_webots_table'))

    # Launch webots
    generate_webots_world_file(robots, source_filename, target_filename)
    webots = WebotsLauncher(world=target_filename)
    launch_description.append(webots)


    # add executables for each robot
    for i in range(N):

        # Guidance
        launch_description.append(Node(
            package='crazychoir_examples',
            executable='crazychoir_task_assignment_webots_guidance'))

        # Simple guidance (takeoff and landing)
        launch_description.append(Node(
            package='crazychoir_examples',
            executable='crazychoir_task_assignment_webots_simple_guidance'))

        # Planner
        launch_description.append(Node(
            package='crazychoir_examples',
            executable='crazychoir_task_assignment_webots_planner'))

        # Webots Driver
        launch_description.append(get_cf_driver(i))
        launch_description.append(Node(
            package='robot_state_publisher',
            executable='robot_state_publisher'))

    return LaunchDescription(launch_description)
```

How to run an Experiment — Video

Scientific Publications

Distributed Feedback Optimization for Multi-robot Target Encirclement and Patrolling
L. Pichierri, G. Carnevale, and G. Notarstefano — *IEEE CASE 2024 (to appear)*

A Tutorial on Distributed Optimization for Cooperative Robotics:
from Setups and Algorithms to Toolboxes and Research Directions
A. Testa, G. Carnevale, and G. Notarstefano — sub. to. Proceedings of the IEEE *2023*

CrazyChoir: Flying Swarms of Crazyflie Quadrotors in ROS 2
L. Pichierri, A. Testa, and G. Notarstefano — *IEEE RAL 2023*

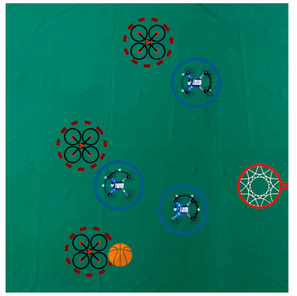
A Distributed Online Optimization Strategy for Cooperative Robotic Surveillance
L. Pichierri, G. Carnevale, L. Sforni, A. Testa, and G. Notarstefano — *IEEE ICRA 2023*

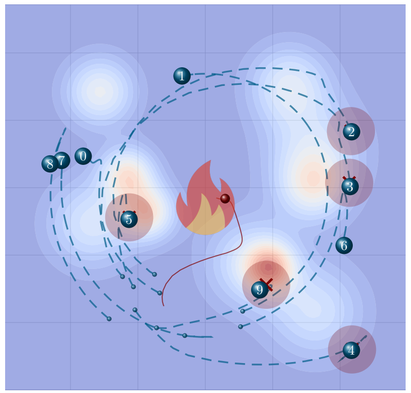
Multi-robot pickup and delivery via distributed resource allocation
A. Camisa, A. Testa, and G. Notarstefano — *IEEE TRO 2022*

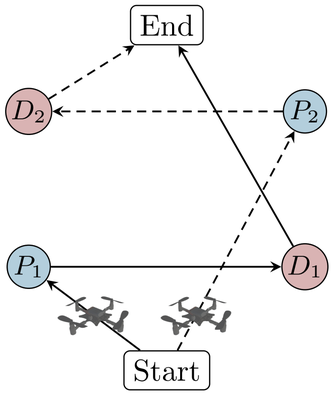
Generalized assignment for multi-robot systems via distributed branch-and-price
A. Testa, and G. Notarstefano — *IEEE TRO 2021*

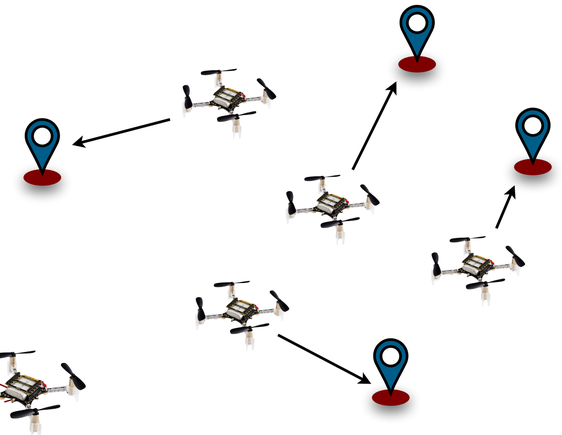
ChoiRbot: A ROS 2 toolbox for cooperative robotics
A. Camisa, A. Testa, and G. Notarstefano — *IEEE RAL 2021*

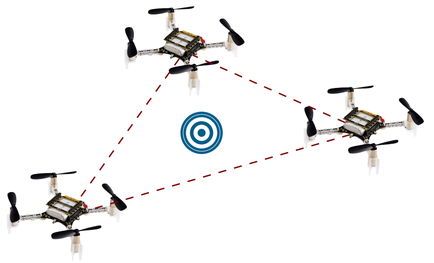
Disropt: a python framework for distributed optimization
F. Farina, A. Camisa, A. Testa, I. Notarnicola, and G. Notarstefano — IFAC-WC 2020











Thanks for your attention!