Multi-Contact Planning
very first draft

Paolo Ferrari

***Abstract*—**

I. Problem Formulation

Possible applications of Multi-Contact Planning and Control (possible simulations/demos)

- reach a desired point with one hand in a cluttered environment (even if walking is not needed)
- stand up from a supine posture
- navigate a rough terrain to reach a desired location, possibly crouching down

Assumptions

- initial "state" $\boldsymbol{q}_{\rm ini}$, $\boldsymbol{F}_{\rm C}^{\rm ini}$ is given
- environment is known and described in the form of a (unique) superquadric S_C
- only quasi-static motions (this is reasonable for safety reasons due to the challenging scenarios that we consider)

Some definitions

- P: set of points of the robot that are allowed to be in contact with the environment. Henceforth, we will just call them end-effectors. For example, $P = \{LFoot, RFoot, LKnee, RKnee, LHand, RHand\}$
- stance $\sigma = \{c_1, \dots, c_{n_\sigma}\}$: a set of contacts with the environment
- contact $c_i = (p_i, \boldsymbol{r}_{{\rm C},i})$: specifies that the end-effector p_i is in contact with the environment in the point located at $\boldsymbol{r}_{{\rm C},i}$
- P_a: set of active end-effectors at a given a stance σ, i.e., those that participate to contacts
- P_n: set of non active end-effectors at a given a stance σ, i.e., those that do not participate to contacts

II. Proposed Approach

The Multi-Contact Planner, whose pseudocode is given in Algorithm 1, searches for a sequence of stances, together with the corresponding contact forces, that allows to complete the assigned task.

The proposed planner builds a tree $\mathcal{T}$ using a RRT-like strategy. In this tree, a vertex $v = (\boldsymbol{q}, \boldsymbol{F}_{\rm C}, P_{\rm a})$ consists in a configuration $\boldsymbol{q}$, a contact forces vector $\boldsymbol{F}_{\rm C}$, and a set of active end-effectors $P_{\rm a}$. This specifies that the robot at configuration $\boldsymbol{q}$, exercising the forces $\boldsymbol{F}_{\rm C}$, is guaranteed to be in static equilibrium and in contact with the environment only with the end-effectors $P_{\rm a}$. Note that, a vertex implicitly specifies a stance (HOW?). An edge between two vertices v and v' indicates that it is possible to move from σ to σ' (respectively, the stances associated to v and v') by either removing or adding a contact.

Algorithm 1: Multi-Contact Planner

```
root the tree 𝒯 at v_ini = (q_ini, F_C^ini, P_a^ini);
j ← 0;
repeat
    randomly pick a sample r_rand ∈ ℝ^3;
    select nearest vertex v_near in 𝒯 to r_rand;
    extract q_near, F_C^near, and P_a^near from v_near;
    P_n^near ← P \ P_a^near;
    randomly choose expansion type f ∈ {remove, add};
    if f = remove then
        randomly pick p_k from P_a^near;
        P_a^cand ← P_a^near \ {p_k};
    else
        randomly pick p_k from P_n^near;
        P_a^cand ← P_a^near ∪ {p_k};
    end
    set r_CoM^des and r_C^des according to f;
    x_cand ← CandidateContacts(r_CoM^des, r_C^des, P_a^cand, p_k, f);
    extract r_CoM^cand, r_C^cand, and F_C^cand from x_cand;
    q_cand ← IKSolution(r_CoM^cand, r_C^cand, F_C^cand);
    if q_cand ≠ ∅ then
        v_new ← (q_cand, F_C^cand, P_a^cand);
        add new vertex v_new to 𝒯 as a child of v_near;
    end
    j ← j + 1;
until SolutionFound() or j > j_max;
if SolutionFound() then
    retrieve sequences S and Q from 𝒯;
    return (S, Q);
end
return ∅;
```

Procedure 1: CandidateContacts($\boldsymbol{r}_{\rm CoM}^{\rm des}$, $\boldsymbol{r}_{\rm C}^{\rm des}$, $P_{\rm a}$, p_k, f)

```
P_n ← P \ P_a;
if f = remove then
    compute x = (r_CoM, r_C, F_C) that solve problem (1);
else
    compute x = (r_CoM, r_C, F_C) that solve problem (2);
end
return x;
```

At the beginning, the tree $\mathcal{T}$ is rooted at vertex $v_{\rm ini} = (\boldsymbol{q}_{\rm ini}, \boldsymbol{F}_{\rm C}^{\rm ini}, P_{\rm a}^{\rm ini})$. Then, the algorithm enters an iterative procedure to construct the tree, whose branches will represent different sequences of stances.

The generic j-th iteration starts by randomly sampling a point $\boldsymbol{r}_{\rm rand}$ in the workspace; the vertex $v_{\rm near}$ in the current tree $\mathcal{T}$ that is the closest to $\boldsymbol{r}_{\rm rand}$ according to a certain distance metric $\gamma(\cdot, \boldsymbol{r}_{\rm rand})$ is selected. In particular, for a generic configuration $\boldsymbol{q}$ and a generic point $\boldsymbol{r}$, we define

Algorithm 2: Joint-space Planner

$i \leftarrow 0$;
repeat
  root tree $\mathcal{T}_q$ at q_i;
  $j \leftarrow 0$;
  repeat
    randomly pick a configuration $\boldsymbol{q}_{\rm rand} \in \mathcal{C}$;
    select nearest vertex $q_{\rm near}$ in $\mathcal{T}_q$ to $\boldsymbol{q}_{\rm rand}$;
    generate a free-flying configuration $\tilde{q}_{\rm new}$;
    compute $q_{\rm new}$ by projecting $\tilde{q}_{\rm new}$ on submanifold $\mathcal{C}_i$;
    if $\boldsymbol{q}_{\rm new}$ is feasible **then**
      add new vertex $\boldsymbol{v}_{\rm new}$ to $\mathcal{T}_q$ as a child of $\boldsymbol{q}_{\rm near}$;
    end
    $j \leftarrow j + 1$;
  until $\boldsymbol{q}_{\rm new} = \boldsymbol{q}_{i+1}$ **or** $j > j_q^{\max}$;
  if $\boldsymbol{q}_{\rm new} = \boldsymbol{q}_{i+1}$ **then**
    retrieve from $\mathcal{T}_q$ the sequence of configurations Q_i;
    compute trajectory $\boldsymbol{q}_i(t)$ interpolating Q_i;
    concatenate $\boldsymbol{q}_i(t)$ to $\boldsymbol{q}(t)$;
  else
    return $\emptyset$;
  end
  delete tree $\mathcal{T}_q$;
  $i \leftarrow i + 1$;
until $i > N$;
return $\boldsymbol{q}(t)$;

such distance metric as the Euclidean distance between the CoM position of the robot at configuration $\boldsymbol{q}$ and $\boldsymbol{r}$, i.e., $\gamma(\boldsymbol{q}, \boldsymbol{r}) = \|\boldsymbol{f}_{\rm CoM}(\boldsymbol{q}) - \boldsymbol{r}\|$. The configuration $\boldsymbol{q}_{\rm near}$, the associated contact forces $\boldsymbol{F}_{\rm C}^{\rm near}$, and the set of active end-effectors $P_{\rm a}^{\rm near}$ are extracted from $v_{\rm near}$, and the set of non active end-effectors is reconstructed as $P_{\rm n}^{\rm near} = P \setminus P_{\rm a}^{\rm near}$.

In general, there exist two possible options for attempting to generate a new stance from the one specified by $v_{\rm near}$ that consist, respectively, in removing or adding a contact from $\sigma_{\rm near}$. The algorithm randomly chooses (f denotes such random choice) between the two options (f = `remove` or f = `add`) with the constraint of respecting the bounds on the number of allowed number contacts. In particular, if only one contact is active at $v_{\rm near}$, i.e., $|P_{\rm a}^{\rm near}| = 1$, then only the addition of a new contact is allowed. On the other hand, if the maximum number of contacts is reached at $v_{\rm near}$, i.e., $|P_{\rm a}^{\rm near}| = m_{\rm C}$, then only the removal of an existing contact is allowed.

At this point, the algorithm decides the identity of the robot end-effector that, according to f, will remove or add a contact. In particular,in the first case (f = `remove`), the end-effector p_k that will be lifted for removing an existing contact is chosen from the set $P_{\rm a}^{\rm near}$; in the second case (f = `add`), the end-effector p_k that will be used for adding a new contact is chosen from the set $P_{\rm n}^{\rm near}$.

The algorithm proceeds in finding the positions of all the end-effectors $\boldsymbol{r}_{\rm C}$, the position of the CoM $\boldsymbol{r}_{\rm CoM}$, and the contact forces $\boldsymbol{F}_{\rm C}$ such that by changing the chosen contact the resulting stance is statically stable. To this end, we employ an adaptation of the optimization-based planner introduced in [RAL2020].

Here, the desired value $\boldsymbol{r}_{\rm CoM}^{\rm des}$ for the CoM position can be set to $\boldsymbol{f}_{\rm CoM}(\boldsymbol{q}_{\rm near})$ to prefer small displacements from the current position, or omitted at all. Instead, the desired value $\boldsymbol{r}_{\rm C}^{\rm des} = (\boldsymbol{r}_{{\rm C},0}^{\rm des}, \dots, \boldsymbol{r}_{{\rm C},n_{\rm C}}^{\rm des})$ for the end-effectors positions are set to $(\boldsymbol{f}_0(\boldsymbol{q}_{\rm near}), \dots, \boldsymbol{f}_{n_{\rm C}}(\boldsymbol{q}_{\rm near}))$, with the exception $\boldsymbol{r}_{{\rm C},k}^{\rm des} = \boldsymbol{r}_{\rm rand}$ in case f = `add`. In the following, we report the formulation of such optimization problems, whose solution is invoked in Procedure 1. [BETTER EXPLAIN HERE]

NLP problem for removing a contact

$$\min_{\boldsymbol{x}} \left\|\boldsymbol{r}_{\rm CoM} - \boldsymbol{r}_{\rm CoM}^{\rm des}\right\|_{2,W_{\rm CoM}}^2 + \left\|\boldsymbol{r}_{{\rm C},k} - \boldsymbol{r}_{{\rm C},k}^{\rm des}\right\|_{2,W_{\rm C}}^2 + \left\|\boldsymbol{F}_{\rm C}\right\|_{2,W_{\rm F}}^2$$

subject to

$$m\boldsymbol{g} + \boldsymbol{G}\boldsymbol{F}_{\rm C} = \boldsymbol{0} \tag{1a}$$
$$\boldsymbol{F}_{{\rm C},i} = \boldsymbol{0}, \quad \forall p_i \in P_{\rm n} \tag{1b}$$
$$\boldsymbol{r}_{{\rm C},i} = \boldsymbol{r}_{{\rm C},i}^{\rm des}, \quad \forall p_i \in P_{\rm a} \tag{1c}$$
$$\boldsymbol{r}_{{\rm C},i}^{\min} \leq \boldsymbol{r}_{{\rm C},i} \leq \boldsymbol{r}_{{\rm C},i}^{\max}, \quad \forall p_i \in P \tag{1d}$$

NLP problem for adding a contact

$$\min_{\boldsymbol{x}} \left\|\boldsymbol{r}_{\rm CoM} - \boldsymbol{r}_{\rm CoM}^{\rm des}\right\|_{2,W_{\rm CoM}}^2 + \left\|\boldsymbol{r}_{{\rm C},k} - \boldsymbol{r}_{{\rm C},k}^{\rm des}\right\|_{2,W_{\rm C}}^2 + \left\|\boldsymbol{F}_{\rm C}\right\|_{2,W_{\rm F}}^2$$

subject to

$$m\boldsymbol{g} + \boldsymbol{G}\boldsymbol{F}_{\rm C} = \boldsymbol{0} \tag{2a}$$
$$\boldsymbol{F}_{{\rm C},i} = \boldsymbol{0}, \quad \forall p_i \in P_{\rm n} \tag{2b}$$
$$\boldsymbol{r}_{{\rm C},i} = \boldsymbol{r}_{{\rm C},i}^{\rm des}, \quad \forall p_i \in P_{\rm a} \backslash \{p_k\} \tag{2c}$$
$$\boldsymbol{r}_{{\rm C},k} \in S_{\rm C}(\boldsymbol{r}_{{\rm C},k}) \tag{2d}$$
$$\{\boldsymbol{F}_{{\rm C},k}, \boldsymbol{r}_{{\rm C},k}\} \in \mathcal{F}(\boldsymbol{F}_{{\rm C},k}, \boldsymbol{r}_{{\rm C},k}, \mu_k) \tag{2e}$$
$$\boldsymbol{r}_{{\rm C},i}^{\min} \leq \boldsymbol{r}_{{\rm C},i} \leq \boldsymbol{r}_{{\rm C},i}^{\max}, \quad \forall p_i \in P \tag{2f}$$

Once the corresponding NLP problem is solved, the algorithm invokes a certain procedure [GOAL SAMPLER ?] that computes a configuration $\boldsymbol{q}_{\rm cand}$ that is collision-free, statically stable and such that the CoM position is at $\boldsymbol{r}_{\rm CoM}^{\rm cand}$ and the end-effectors are at the positions specified in $\boldsymbol{r}_{\rm C}^{\rm cand}$. If such configuration exists, a new vertex is created as $v_{\rm new}$ = ($\boldsymbol{q}_{\rm cand}$, $\boldsymbol{F}_{\rm C}^{\rm cand}$, $P_{\rm a}^{\rm cand}$), with $P_{\rm a}^{\rm cand} = P_{\rm a}^{\rm near} \setminus \{p_k\}$ or $P_{\rm a}^{\rm cand} = P_{\rm a}^{\rm near} \cup \{p_k\}$ depending if the contact with end-effector p_k has been removed or added.

The described iterative procedure terminates when a solution for the planning problem is found, or when a fixed maximum number of expansions is exceeded. In the first case, the sequence of stances $S = (\sigma_0, \dots, \sigma_N)$, and the sequence of associated statically-stable configurations $Q = (\boldsymbol{q}_0, \dots, \boldsymbol{q}_N)$ are retrieved and passed to the Joint-space Planner [IROS2020], whose sketch is provided in Algorithm 2.

References