H2020 RIA
COMANOID
H2020-RIA-645097

Deliverable D3.3:
Final safety guidelines and strategies

March 15, 2019
The research leading to these results has received funding from the European Union H2020 Program/2015-2020 under grant agreement n°645097 COMANOID.
## Contents

1 Preliminaries ........................................ 4
   1.1 Mission environments .......................... 4
   1.2 Robot contexts ................................ 4
   1.3 General safety guidelines ...................... 5

2 Safety Behaviors ..................................... 6
   2.1 Assumptions .................................... 6
   2.2 Overview ...................................... 7
      2.2.1 Override behaviors ....................... 7
      2.2.2 Temporary override behaviors ............ 8
      2.2.3 Proactive behaviors ...................... 8
   2.3 Safety areas and thresholds .................... 9
   2.4 Behavior specifications ....................... 10
   2.5 State Machine ................................ 14

3 Implementation ..................................... 17
   3.1 Control scheme ................................ 17
      3.1.1 Head task generator ..................... 17
      3.1.2 Locomotion task generator ............... 17
   3.2 Safety behavior blocks ....................... 18
      3.2.1 halt ........................................ 18
      3.2.2 self-preserve ............................ 19
      3.2.3 stop ........................................ 19
      3.2.4 evade ..................................... 19
      3.2.5 scan and track ............................ 19
      3.2.6 adapt footsteps .......................... 19
   3.3 Simulations ................................... 20
Chapter 1
Preliminaries

The objective of this chapter is to introduce some basic concepts (mission environments and robot contexts) that are needed to discuss the activation of safe behaviors. Also, we provide a qualitative description of the safety guidelines that have inspired the design of such behaviors.

Mission environments

The mission environments identify the spatial zones where the robot must operate in the various phases of the demo mission. We define three mission environments:

- **Flat floor**: The robot is moving on the ground floor or on the first floor landing (flat surfaces). On the ground floor, the robot is going to or coming from the stairs. On the first floor, it is approaching the location where the bracket is, or moving from the latter location towards the entrance of the cargo area, or returning from the cargo area towards the stairs.

- **Stairs**: The robot is climbing (descending) the stairs to go from the ground (first) floor to the first (ground) floor. The stairs are equipped with a handrail.

- **Cargo area**: The robot is moving inside the cargo area for the main part of its mission.

Robot contexts

The robot contexts characterize what the robot is doing at a certain time instant. We identify five robot contexts:

- **Idle**: The robot is standing in double support at a fixed position and not performing any particular task.

- **Manipulation**: The robot is standing in double support and it is executing a task (typically a manipulation task) that does not require any stepping.

- **Locomotion**: The robot is moving in the environment by taking steps. This includes walking, multi-contact locomotion and stair climbing/descending.
The robot has lost balance and is falling.

- **Fall.** The robot is on hold until restarted by human intervention.

The first three are normal operation contexts, whereas the last two are emergency contexts. In particular, an **Idle** context can be included in a mission plan as an intentional pause, e.g., at the transition between different phases or for debugging purposes. **Error** will be the release context for behaviors that are associated to a failure (halt, self-protect).

### General safety guidelines

We now introduce some general guidelines that should be followed for safe robot operation.

- **Watch what you’re doing.** The robot should always watch its main area of operation. When performing a manipulation task, it will therefore look at its hand or at the object to be manipulated. When walking, it should keep its gaze directed to the area where it is about to step.

- **Be on the lookout.** If the robot is idle, then it can freely monitor the environment, focusing in particular on objects that are not included in its current 3D map (e.g., moving objects).

- **Evade if you can.** When a moving object approaches the robot, perform an evasive action if this can be done safely.

- **Stop if you must.** In a situation of clear and present danger, stop any operation as soon as possible.

- **Respect humans.** In the presence of humans, robot velocities and forces should be scaled down in order to reduce the potential damage in the case of a collision [ISO TS 15066, 2016].

- **Look for support.** When locomotion is expected to be challenging (e.g., on the stairs), the robot should try to establish additional contact with the environment (e.g., with the handrail), so that it has at least two support points at all times. The possibility of improving balance by adding contacts should also be considered whenever a significant risk of falling is detected.

- **Preserve yourself.** In the imminence of a potentially damaging event, such as an unavoidable fall, the robot should act so as to minimize damage to itself and the environment.

Some of these guidelines will be realized as safety behaviors (see next chapter) that are activated to improve the level of safety in dangerous situations, while others must also be taken into account within the planning/control stage. For example, **watch what you’re doing** calls for visual-servoed manipulation or locomotion strategies; **look for support** has consequences at the planning/stage (e.g., generation of stair climbing motions must include handrail grasping and releasing) but will also result in a safety behavior.
Chapter 2
Safety Behaviors

We now describe the behaviors that the robot adopts to guarantee safety for itself and the environment (including humans). First, we discuss the assumptions under which these behaviors have been designed, particularly in terms of which information must be available for their activation, and the associated definitions of safety areas and thresholds.

Assumptions

We assume that at any time instant:

- The current mission environment is one of those defined in Section 1.2, and the robot is in one of the contexts defined in Section 1.2.
  For example, the robot may be Idle on flat floor, or in Locomotion in the cargo area. The other assumptions are related to which information must be made available at all instants by the sensory system. We do not discuss in any detail the perception processes that provide such information.
- The robot is aware of its location w.r.t. a 3D map maintained by the SLAM module.
- The robot is aware of the current risk of fall $r_{\text{fall}}$. Estimation of $r_{\text{fall}}$ will be based on inertial measurements, and should consider the ZMP support area [Caron and Kheddar, 2016, Caron et al., 2017] as well as the context in which the robot is operating (e.g., risk of fall in the stairs environment should be intrinsically larger than on flat floor).
- The robot is aware of the current battery level $l_{\text{battery}}$.
- The robot is made aware of contact surfaces in its workspace through flag $f_{cs}$. Contact surfaces are surfaces (or points) of the 3D map with which the robot may safely establish a contact for additional support. Labeling of contact surfaces is made at the SLAM level.
- The robot is made aware of unexpected objects in a proximal area $P$ whose extension depends on the specific sensory system. In particular, it knows the minimum distance $d_{uo}$ to the closest unexpected object. Unexpected objects are objects that are not present in the current 3D map, and may be moving (e.g., humans) or not (e.g., cables on the floor). If there is no such object in $P$, $d_{uo}$ is set to $\infty$. 

6
Figure 2.1: The activation of safety behaviors depends on the current mission environment and robot context, and is triggered by information coming from the sensory system.

- The robot is made aware of unexpected contacts with the environment through flag $f_{uc}$. Depending on the contact detection mechanism, other information may be available, such as the location of the contact point in terms of the robot kinematic chains.

Overview

We introduce three kinds of safety behaviors:

1. **Override** behaviors will stop the execution of the current task and lead to a state from which normal operation can only be resumed only by means of human intervention.

2. **Temporary override** behaviors will also suspend task execution, but only for a limited amount of time. As soon as safety concerns have been properly handled, task execution will resume automatically.

3. **Proactive** behaviors, on the other hand, do not stop the task, but try to increase the overall safety level by calling for an adaptation or enhancement of the current robot activity.

This section will briefly describe the idea behind each behavior, and the situations in which it will be activated. A more specific description, with detailed triggering conditions for each case, will be given in the rest of the chapter.

Override behaviors

Override behaviors will interrupt the current robot operation and lead to a state from which it cannot be resumed without human intervention. They are intended as a way to react to unexpected and dangerous situations from which it is unclear whether the task can be safely resumed automatically. We define two override behaviors, namely *halt* and *self-preserve*.

- **halt**: Many situations that interfere with safe operations may occur (e.g. the battery level is too low or a moving object is getting too close to the robot area of operation). In that case,
following the stop if you must guideline, the robot must be ready to immediately interrupt the execution of its tasks. The halt behavior is an emergency stop procedure which takes care of interrupting any operation, keeping in mind that a humanoid robot, especially during locomotion, needs to stop in a way that maintains balance.

• self-preserve: Although it would be desirable to avoid falls altogether, there are several reasons which might lead to a loss of balance, for example a hardware/software fault, or an external perturbation (e.g., a collision with an unseen object). To properly handle this, a self-preserve behavior to be activated during falls can be designed. When the robot detects an unrecoverable loss of balance, it must immediately stop any task execution and adopt measures aimed at minimizing damage to itself and the environment due to the fall.

Temporary override behaviors

Some events which might cause safety concerns may require to stop task execution only for a limited amount of time. This is the case of moving obstacles entering the robot safety area (e.g., humans or other robots).

• stop: an unexpected moving obstacle at a certain distance from the robot does not constitute an immediate threat, but it might do if it gets closer. In order be ready to take action the robot suspends any locomotion task and stops. If the unexpected obstacle leaves the robot safety area, the locomotion task can resume automatically. This behavior differs from halt which requires human intervention to restart the robot.

• evade: following the evade if you can guideline, if the unexpected moving obstacle gets sufficiently close to the robot, the latter performs an evasive maneuver to avoid the collision. At the end of the maneuver the robot can resume normal operations if the obstacle does not constitute a threat anymore.

Proactive behaviors

Proactive behaviors are actions intended to modify the current robot task in order to increase its overall safety, without interrupting the main task.

• track: Whenever possible without interfering with the current task, the robot keeps its gaze directed at the closest unexpected object (e.g. something that is not in the local robot map) in its proximal area, as suggested by the be on the lookout guideline.

• adapt_footsteps: during locomotion, the robot is in general controlled via a high-level command, such as tracking a reference velocity, or reaching a specific location in the workspace. The adapt_footsteps behavior allows the robot to modify its footstep plan to avoid collision with unexpected objects in its path.

• scale_velocity-force: during manipulation the robot must be able to coexist with other agents in the environment. For this reason, according to the guideline respect humans, we introduce a behavior which will scale velocities or forces associated to the current manipulation task. If a close unexpected object is perceived, the robot reduces its velocities/forces to reduce the risk of injures in case a collision occurs.
Figure 2.2: The relationships between the different thresholds used for quantities \( d_{uo} \) (top) and \( r_{fall} \) (bottom), each with its corresponding behaviors.

- **add_contact**: the add_contact behavior, following the look for support guideline is a procedure that allows the robot to try to establish new contacts for additional support, if this can be done without interfering with the current task.

**Safety areas and thresholds**

Most safety behaviors are triggered by some value being higher or lower than some given threshold. This can be the distance of the robot from an unexpected obstacle \( d_{uo} \) or the estimated risk of fall \( r_{fall} \).

Figure 2.2, top, summarizes the different thresholds that have been defined on \( d_{uo} \): \( d_{halt}^{track} \), \( d_{scale}^{track} \), \( d_{evade}^{track} \), \( d_{adapt}^{track} \), and \( d_{halt}^{track} \). These thresholds implicitly define a set of concentric annular areas, shown in Fig. 2.3, around the robot:

- **\( S^{track} \)** - visual tracking area: this is the area defined by \( d_{halt}^{track} < d_{uo} < d_{track}^{track} \). If the unexpected moving obstacle enters this area the robot should be tracking if \textit{Idle}. If in \textit{Locomotion} it should first stop and then start tracking.

- **\( S^{adapt} \)** - footstep adaptation area: this is the area defined by \( d_{halt}^{adapt} < d_{uo} < d_{adapt}^{adapt} \). If an unexpected fixed obstacle is in this area the robot should adapt its footsteps to avoid collision with it.

- **\( S^{evade} \)** - evasion area: this is the area defined by \( d_{halt}^{evade} < d_{uo} < d_{evade}^{evade} \). If the unexpected moving obstacle enters this area and the robot is \textit{Idle}, it should plan an evasion maneuver and execute it.

- **\( S^{scale} \)** - velocity/force scaling area: this is the area defined by \( d_{halt}^{scale} < d_{uo} < d_{scale}^{scale} \). If the robot enters \textit{Manipulation} and an unexpected moving obstacle enters this area, the robot should reduce the velocities and forces associated to the manipulation task to avoid damaging contacts with the obstacle.

- **\( S^{halt} \)** - halt area: this is the area defined by \( d_{uo} < d_{halt}^{halt} \). This is the innermost safety area. If any unexpected obstacle enters this area the robot must safely stop whatever action it is performing and request human intervention.
Figure 2.3: Safety areas assuming $d_{\text{evade}} = d_{\text{adapt}}$.

Figure 2.2, bottom, shows the thresholds used for the risk of fall $r_{\text{fall}}$. As soon as $r_{\text{fall}}$ becomes significant ($r_{\text{low fall}} < r_{\text{fall}} < r_{\text{high fall}}$) the robot activates the add contact behavior. If a fall is deemed inevitable ($r_{\text{fall}} > r_{\text{high fall}}$), the robot will perform a safe fall.

**Behavior specifications**

We will now present a detailed description of each safety behavior, which will include:

- the contexts in which the behavior can be activated;
- a trigger, which indicates what particular event or piece of information will cause the behavior to activate, along with specific actions that occur upon triggering, such as deactivating other behaviors;
- a release, which is an event or piece of information that causes the behavior to deactivate, also including specific actions that occur upon deactivation;
- the action, which is what the behavior actually does as long as it is active.

All triggering conditions are continuously checked by the robot. If its triggering condition is met, a behavior can therefore deactivate another behavior prior to its natural release. For example a halt may occur at any time during normal robot operation, deactivating all active behaviors.

**scan**

- **Context**: *Idle*, *Manipulation*, *Locomotion*
- **Trigger**: active by default whenever the *track* behavior is inactive
- **Release**: no release condition
This is the default behavior during task execution, when no special safety concerns are present. This behavior can be activated in any of the normal operation contexts, which results in the following three states of normal operation: *Idle/scan* in which the robot is standing in a fixed position while scanning as much as possible of the surrounding environment, *Locomotion/scan* in which the robot is assigned a locomotion task and it is scanning the path ahead to detect any unexpected obstacle, *Manipulation/scan* in which the robot is standing while performing a manipulation task, and thus scanning the area of operation.

Note that although there is no release condition, *scan* will be deactivated when *track* is activated, as the robot can at any time either scan an area or track a particular object.

**self-protect**

- **Context**: *Idle, Manipulation, Locomotion, Error*
- **Trigger**: fall detected \(r_{\text{fall}} < r_{\text{high}}\) upon triggering changes the context to *Fall* and deactivates all other behaviors
- **Release**: when fallen upon release the context is changed to *Error*
- **Action**: the robot acts so as to minimize the potential damage to its own structures and/or the environment. To this end, several aspects must be considered (see D3.2: Safety falling procedures and strategies) including (i) how to fall, i.e., which internal posture to assume before impact to preserve robot integrity (ii) where to fall, i.e., how to choose the landing surfaces so as to avoid fragile components.

The *self-protect* behavior is activated when the robot is unable to keep balance, and an unavoidable fall is detected. This will start a procedure to minimize damage to the robot and the surrounding environment. It is an override behavior, so upon triggering it deactivates any other behavior.

**halt**

- **Context**: *Idle, Manipulation, Locomotion*
- **Trigger**: three events may independently trigger an emergency stop:
  - E1: an unexpected obstacle enters \(S_{\text{halt}}\)
  - E2: \(f_{\text{uc}}\) is active (unexpected contact);
  - E3: \(l_{\text{battery}} < l_{\text{low}}\), where \(l_{\text{battery}}\) is the minimum acceptable battery level for safe operation.

  upon triggering deactivates any other behavior
- **Release**: when the robot is fully stopped upon release changes the context to *Error*
• **Action**: the specific action depends on the triggering conditions:
  
  – **A1**: If the context was *Idle* and the triggering event was E1 or E3, the robot will augment its support polygon and/or assume a low-impact configuration (e.g., by folding its arms);
  – **A2**: If the context was *Idle* and the triggering event was E2, the robot will immediately decrease joint stiffness on the kinematic chain where the contact has occurred, provided that the latter is on the upper body. Otherwise, the robot will maintain its current pose;
  – **A3**: If the context was *Manipulation* the robot should safely abort the task and stop its motion as soon as possible, regardless of the triggering event;
  – **A4**: If the context was *Locomotion*, the robot should stop walking as soon as possible, regardless of the triggering event.

The *halt* behavior is an override, so upon triggering it deactivates any other behavior, and also inhibits any behavior from activating except for *self-protect* in case of a fall.

**stop**

• **Context**: *Locomotion*

• **Trigger**: an unexpected moving obstacle enter $S_{\text{track}}$ upon triggering deactivates any other behavior

• **Release**: this behavior is released when the robot is fully stopped, at which point the context is changed to *Idle*

• **Action**: stops the robot in a double support configuration

The robot stops if a moving unexpected obstacle comes too close. This behavior is activated in preparation for the execution of an evasive maneuver if the obstacle gets closer to the robot.

**evade**

• **Context**: *Idle*

• **Trigger**: an unexpected moving obstacle enters $S_{\text{evade}}$ upon triggering changes context to *Locomotion*

• **Release**: the unexpected moving obstacle leaves $S_{\text{evade}}$ upon release changes context to *Idle*

• **Action**: an evasion maneuver is planned in real time based on the spatial relationship between the robot and the approaching object [Cognetti et al., 2016]. Feasibility of the maneuver with respect to the current 3D map is continuously checked. Whenever the maneuver becomes unfeasible, the associated flag $f_{\text{evade}}$ is set to FALSE and the halt behavior is activated.
If an unexpected object approaches the robot in the *idle* state, the latter should execute an
evasive maneuver, provided this can be done safely (*evade if you can* guideline).

Note that *evade* upon activation changes the context to *Locomotion*, because it requires the
robot to be taking steps.

*Evade* cannot be activated if the *stop* behavior is active.

**track**

- **Context**: *Idle*

- **Trigger**: an unexpected obstacle enters $S_{\text{track}}$
  upon triggering deactivates *scan*

- **Release**: the unexpected obstacle leaves $S_{\text{track}}$
  upon release activates *scan*

- **Action**: the robot should move its head so as to track the closest unexpected obstacle in
  its *proximal area*.

If the robot is in *Idle* and an unexpected object gets close enough to enter the safety area $S_{\text{track}}$, the robot will direct its gaze at it (*be on the lookout* guideline). Note that this behavior cannot
be triggered if the robot is in any *manipulation* state. In these cases, in fact averting the gaze
from the current task can be dangerous (*watch what you’re doing* guideline).

Note that if in *Idle*, the robot will simply start tracking the obstacle, while if in *Locomotion* it
needs to *stop* first. After stopping the robot is in *Idle*, at which point *track* will be activated if
the trigger condition is still met.

**adapt_footsteps**

- **Context**: *Locomotion*

- **Trigger**: a static obstacle is inside $S_{\text{adapt}}$

- **Release**: the static obstacle is outside $S_{\text{adapt}}$

- **Action**: the robot adapts the planned footsteps to take into account unexpected fixed
  obstacles in the scene

During locomotion, it is possible that unexpected objects (such as cables on the ground) may
interfere with the planned footsteps. In this case, the robot should modify the planned footsteps
to avoid collision with the object.
**scale_velocity-force**

- **Context**: Manipulation
- **Trigger**: an unexpected moving obstacle enters $S_{\text{scale}}$
- **Release**: the unexpected moving obstacle leaves $S_{\text{scale}}$
- **Action**: during manipulation, robot velocities and/or forces need to be reduced, following the *respect humans* guideline

**add_contact**

- **Context**: Idle, Manipulation
- **Trigger**: $r_{\text{fall}}^{\text{low}} < r_{\text{fall}} \leq r_{\text{fall}}^{\text{high}}$ AND ($f_{cs} = \text{TRUE}$)
- **Release**: $r_{\text{fall}} > r_{\text{fall}}^{\text{high}}$
- **Action**: the robot selects an additional support point and establishes contact

In the presence of a moderate risk of fall, the robot tries to establish new contacts for additional support (*look for support* guideline).

Here, $r_{\text{fall}}^{\text{low}}$ is the threshold above which the risk of fall is considered to be moderate. Note the role of flag $f_{cs}$ which indicates the presence of a reachable contact surface in the robot workspace. *Locomotion* is not a triggering state for this behavior because it is intrinsically risky to try to establish a new contact while the robot is walking. Moreover, if the robot is climbing/descending stairs, additional contact with the handrail has already been enforced at the planning stage.

**State Machine**

As already noted, the system can be represented as a state machine in which the state of the robot is uniquely identified by the current context and all active behaviors. This will be denoted as $\text{Context}/\text{behavior}_1/\text{behavior}_2$, if for example two behaviors are active.

A complete list of the states is given by:

- Idle/scan
- Idle/scan/add_contact
- Idle/track
- Idle/track/add_contact
- Idle/halt
- Locomotion/scan
- Locomotion/scan/adapt_footsteps
- Locomotion/scan/stop
- Locomotion/track/evasion
- Locomotion/track/evasion/adapt_footsteps
- Locomotion/halt
- Manipulation/scan
- Manipulation/scan/add_contact
- Manipulation/scan/scale_velocity-force
- Manipulation/scan/scale_velocity-force/add_contact
- Manipulation/halt
- Fall/self-protect
- Error

Figure 2.4 gives a simplified representation of the state machine. Transition from one state to another correspond to activation or deactivation of some behavior.

Some states were omitted in the scheme to obtain a more compact representation, and thus by only showing a subset of the possible states and transitions. Of course, the complete scheme can be represented as every behavior is uniquely defined by its trigger and release conditions.
Figure 2.4: A partial representation of the state machine governing the relationships among the proposed safety behaviors. For compactness all states with more than one behavior were omitted, some behavior names were shortened and the contexts are denoted only by their initial (e.g., I for Idle, M for Manipulation, ...). The asterisk * indicates that any other state can go to Fall/self-protect.
Chapter 3
Implementation

The implementation of the safety behavior architecture will invariably depend on the specific platform and control architecture. Here we will refer to a generic architecture for illustration.

Control scheme

Figure 3.1 shows a general overview of the control scheme. As already noted, some details are intentionally left unspecified as they can be implemented in different ways, which may not be relevant for safety purpose (e.g., control of the robot head).

Notice that, while most behaviors can be realized with the proposed scheme by appropriately activating and deactivating the relevant blocks, the self-preserve behavior will need a separate controller which overrides the scheme in Fig. 3.1.

Head task generator

The Head Task Generator is in charge of generating suitable control inputs for the robot head based on information available through vision. Its input will basically switch between two possible behaviors which are scan and track.

Locomotion task generator

We consider a locomotion task generator based on Model Predictive Control (MPC). In particular, the proposed scheme [Scianca et al., 2019] consists of two sequential modules running in real time: the first generates candidate footstep locations based on the high-level motion commands and the environment map, while the second produces a trajectory of the robot CoM and feet such that balance is guaranteed at all times (by ensuring that the ZMP is at all times within the support polygon). Both modules require the solution of a QP problem. In particular, the second module includes a stability constraint (or, correspondingly, a terminal constraint) that is needed to avoid divergence of the CoM with respect to the ZMP; for this reason, it is called Intrinsically Stable MPC (IS-MPC).

The proposed scheme requires — and makes most use of — preview information on the high-level velocity commands. This leads to a locomotion controller that is suitable not only for regular
There is also a concern regarding feasibility, i.e., the ability of the controller to find a solution under the imposed constraints. In [Scianca et al., 2019], IS-MPC was shown to be recursively feasible with a proper choice of the preview horizon, which means that feasibility at any given iteration implies feasibility at the next iteration.

The MPC scheme automatically places footsteps, matching them as much as possible to those provided by the footstep planner. This means it can react to disturbances by adjusting the footstep position, and it will also turn to be useful in the adapt_footsteps block.

Safety behavior blocks

This section will provide brief pointers for the implementation of the safety behavior blocks.

The scalevelocities-forces and add_contact behaviors are not discussed. The first simply requires a procedure for decreasing workspace velocities during manipulation, until the unexpected object leaves the $S^{scale}$ area. As for the second, some preliminary ideas are given in D3.2: Safety falling procedures and strategies.

halt

The halt behavior defines a procedure for interrupting any operation and bringing the robot to a statically balanced position (i.e., CoM and ZMP are coincident and inside the support polygon).
This can be achieved by immediately setting to zero the high-level velocity commands, and changing the ZMP constraint from an alternating single support constraint (which dictates the gait) to a static double support constraint. IS-MPC will generate a trajectory which leads the robot to a halt. This also requires using the appropriate terminal constraint (i.e., the capturability constraint) in order to preserve feasibility, as discussed extensively in [Scianca et al., 2019].

Another possibility relies on the concept of capture point [Pratt et al., 2006] which is the point where the robot should step in order to reach a complete stop. A procedure can be designed by considering whether the capture point is already inside the support polygon (0-step capturability) so that the robot can simply stop in place, or if it is reachable by stepping (1-step capturability), in which case a further step must be planned and executed before reaching a halt.

**self-preserve**

As already mentioned, the self-preserve behavior completely overrides the control architecture presented in Fig. 3.1, which is designed for safely executing locomotion and manipulation tasks. The design of a controller for this behavior can be based on the ideas discussed in D3.2: Safety falling procedures and strategies.

**stop**

Stop is achieved by setting the reference velocity to an appropriate profile that goes to zero in a reasonable time. At the end, the structure of the balance constraints inside the MPC framework is changed, so that it enforces a stationary double support constraint in place of the moving constraints that correspond to automatic footstep placement. Again, the capturability terminal constraint must be used to guarantee feasibility.

**evade**

The evade behavior is realized by temporarily overriding the robot high-level velocity commands in order to avoid an unexpected moving obstacle. The new commands that realize an evasion maneuver are described in [De Simone et al., 2017]. The maneuver will end as soon as the obstacle leaves the area $S_{\text{evade}}$.

**scan and track**

These behaviors are implemented through an appropriate definition of a visual target on the robot camera image plane. In particular, scan will require an artificial target whose motion scans the area of interest, whereas in track the target is obviously the (closest point on the) unexpected object. Visual servoing of the target is achieved using standard techniques [Chaumette and Hutchinson, 2006].

**adapt_footsteps**

During normal operation, the Candidate Footsteps Generator provides the IS-MPC block with reference footstep positions, which are computed based on the available map. If a static unexpected obstacle in $S_{\text{adapt}}$ blocks the path, the adapt_footsteps behavior is invoked to adapt the footsteps so as to avoid the obstacle. This is achieved as described in [De Simone et al., 2017], which takes
Figure 3.2: Snapshots from a simulation of the complete safety framework into account the presence of the unexpected obstacle at two levels: in the cost function of the footstep generation QP, and by the introduction of a collision avoidance constraint in IS-MPC.

Simulations

Figure 3.2 shows a simulation of the safety behavior framework on the HRP-4 humanoid robot.

The robot starts in \textit{Idle/scan} and thus it is moving its head in order to scan the environment with the on-board camera. As soon as the human enters $S^{\text{track}}$ (light red area), the robot detects an unexpected moving object and activates the \textit{track} behavior and stops scanning. The human then enters $S^{\text{evade}}$ (dark red area) which triggers the \textit{evade} behavior, and the state is changed to $\text{Locomotion/track/evade}$. The maneuver ends when the human leaves $S^{\text{evade}}$, at which point the robot goes back to tracking, and then to scanning when the human leaves $S^{\text{track}}$.

In the second part of the simulation the robot is commanded to reach a position in the environment, so the state is changed to $\text{Locomotion/scan}$. It encounters an unexpected fixed obstacle on its path, which triggers the activation of the \textit{adapt_footsteps} behavior. After avoiding the obstacle the robot reaches the goal and stops there, at which point the state is set to \textit{Idle/scan}. 

20
Cited Works


