Middleware-soccer

Generated by Doxygen 1.8.14

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

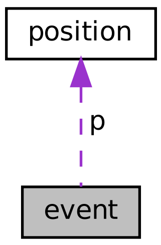
Data Structure Documentation

3.1 event Struct Reference

Event from sensor.

```
#include <common.h>
```

Collaboration diagram for event:



Data Fields

- sid_t **sid**
- picoseconds **ts**
- position **p**

3.1.1 Detailed Description

Event from sensor.

Each event is characterized by:

- the id of the sensor which has generated it,
- a timestamp,
- the registered position.

Definition at line 94 of file common.h.

The documentation for this struct was generated from the following file:

- include/common.h

3.2 interruption_event Struct Reference

Interruption event.

```
#include <common.h>
```

Data Fields

- picoseconds **start**
- picoseconds **end**

3.2.1 Detailed Description

Interruption event.

Each interruption_event is characterized by: the timestamps of beginning and end of the interruption. During an interruption event statistics are updated.

Definition at line 106 of file common.h.

The documentation for this struct was generated from the following file:

- include/common.h

3.3 position Struct Reference

Position coordinates in the game field.

```
#include <common.h>
```

Data Fields

- int32_t **x**
- int32_t **y**
- int32_t **z**

3.3.1 Detailed Description

Position coordinates in the game field.

x, y, z describe the position of the sensor in mm and the origin is the middle of a full size football field.

Definition at line 80 of file common.h.

The documentation for this struct was generated from the following file:

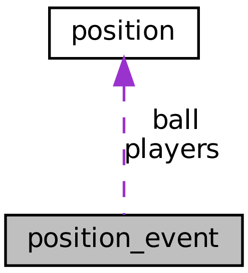
- include/common.h

3.4 position_event Struct Reference

It show a game snapshot.

```
#include <common.h>
```

Collaboration diagram for position_event:



Data Fields

- position **players** [17]
- position **ball**
- int32_t **interval_id**

3.4.1 Detailed Description

It show a game snapshot.

It is characterized by:

- an array with every player position,
- the ball position,
- the specific id of the interval in which the snapshot was taken.

Definition at line 119 of file common.h.

The documentation for this struct was generated from the following file:

- include/common.h

Chapter 4

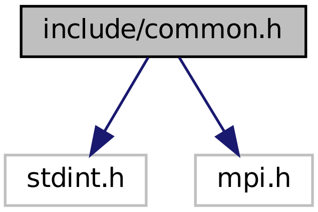
File Documentation

4.1 include/common.h File Reference

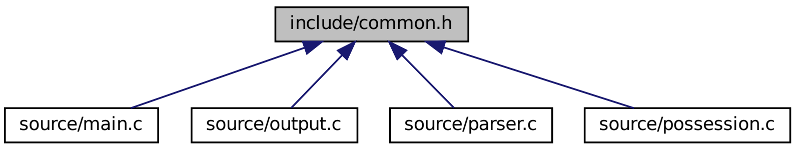
Common constants and definitions.

```
#include <stdint.h>
#include <mpi.h>
```

Include dependency graph for common.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct position

  Position coordinates in the game field.
- struct event

  Event from sensor.
- struct interruption_event

  Interruption event.
- struct position_event

  It show a game snapshot.

Macros

- #define **PRGDEBUG** 0
- #define **DBG**(x) /∗nothing∗/
- #define **FULLGAME_PATH** "../datasets/full-game"
- #define **FIRST_INTERRUPTIONS** "../datasets/referee-events/Game Interruption/1st Half.csv"
- #define **SECOND_INTERRUPTIONS** "../datasets/referee-events/Game Interruption/2nd Half.csv"
- #define **XMIN** 0
- #define **XMAX** 52483
- #define **YMIN** (-33960)
- #define **YMAX** 33965
- #define SECTOPIC 1000000000000
- #define **GAME_START** 10753295594424116
- #define **FIRST_END** 12557295594424116
- #define **SECOND_START** 13086639146403495
- #define **GAME_END** 14879639146403495
- #define **PARSER_RANK** 0
- #define **OUTPUT_RANK** 1
- #define **POSSESSION_RANK** 2
- #define **POSITIONS_MESSAGE** 0
- #define **PRINT_MESSAGE** 1
- #define **POSSESSION_MESSAGE** 2
- #define **ENDOFGAME_MESSAGE** 3
- #define **POSSESSION_BUFFER_SIZE** 1
- #define **IGNORE_GOALKEEPER** 0

Typedefs

- typedef uint32_t sid_t

  Sensor id type.
- typedef uint32_t player_t

  Player type.
- typedef uint64_t **picoseconds**
- typedef struct position position

  Position coordinates in the game field.
- typedef struct event event

  Event from sensor.
- typedef struct interruption_event interruption_event

  Interruption event.
- typedef struct position_event position_event

  It show a game snapshot.

Enumerations

- enum sensor_type_t { **PLAYER**, **REFEREE**, **BALL**, **NONE** }

  Each sensor register data from a specific PLAYER, from the REFEREE or from the BALL; NONE applied when none of the previous case holds.

4.1.1 Detailed Description

Common constants and definitions.

This file contains type definitions and global constants used by processes.

4.1.2 Macro Definition Documentation

4.1.2.1 SECTOPIC

```
#define SECTOPIC 1000000000000
```

Conversion factor from seconds to picosends.

Definition at line 37 of file common.h.

4.1.3 Typedef Documentation

4.1.3.1 event

```
typedef struct event event
```

Event from sensor.

Each event is characterized by:

- the id of the sensor which has generated it,
- a timestamp,
- the registered position.

4.1.3.2 interruption_event

```
typedef struct interruption_event interruption_event
```

Interruption event.

Each interruption_event is characterized by: the timestamps of beginning and end of the interruption. During an interruption event statistics are updated.

4.1.3.3 position

```
typedef struct position position
```

Position coordinates in the game field.

x, y, z describe the position of the sensor in mm and the origin is the middle of a full size football field.

4.1.3.4 position_event

```
typedef struct position_event position_event
```

It show a game snapshot.

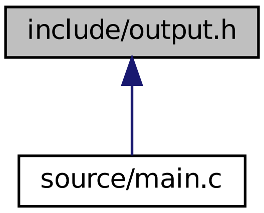
It is characterized by:

- an array with every player position,
- the ball position,
- the specific id of the interval in which the snapshot was taken.

4.2 include/output.h File Reference

output.c function declaration.

This graph shows which files directly or indirectly include this file:



Functions

- void output_run (MPI_Datatype mpi_output_envelope, picoseconds T)

  Core of output's job.
- void print_interval (int interval, picoseconds T)

  It prints the interval header with the current game time.
- void print_statistics (const unsigned int ∗interval_possession, const unsigned int ∗total_possession, int interval, picoseconds T)

  It prints for every team and every member last interval statistic, followed by current cumulative statistics.

4.2.1 Detailed Description

output.c function declaration.

4.2.2 Function Documentation

4.2.2.1 output_run()

```
void output_run (
            MPI_Datatype mpi_output_envelope,
            picoseconds T )
```

Core of output's job.

It keeps waiting for a PRINT_MESSAGE or a POSSESSION_MESSAGE, from onevent or possession processes, until receiving the END_OF_GAME message. After receiving a POSSESSION_MESSAGE, statistics are updated; after receiving a PRINT_MESSAGE, interval and cumulative statistics are printed, by calling print_statistics method, and interval ones are reset; after receiving the END_OF_GAME message, the process exits, after waiting for any pending request. If the received message is of any other type, the process abort.

=1mm

spread 0pt [l]|X[-1,l]|X[-1,l]|**Parameters**

Parameters

mpi_output_envelope	mpi_datatype of received messages.

Definition at line 186 of file output.c.

4.2.2.2 print_interval()

```
void print_interval (
            int interval,
            picoseconds T )
```

It prints the interval header with the current game time.

=1mm

spread 0pt [l]|X[-1,l]|X[-1,l]|**Parameters**

Parameters

interval Current interval id.

T Interval length (in picoseconds).

Definition at line 53 of file output.c.

4.2.2.3 print_statistics()

```
void print_statistics (
            const unsigned int * interval_possession,
            const unsigned int * total_possession,
            int interval,
            picoseconds T )
```

It prints for every team and every member last interval statistic, followed by current cumulative statistics.

=1mm

spread 0pt [l]|X[-1,l]|X[-1,l]|**Parameters**

Parameters

interval_possession Array with last interval statistics for every player (each identified by a constant position in the array).

total_possession Array with cumulative statistics for every player (each identified by a constant position in the array).

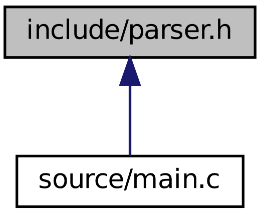
interval Incrementing value used to identify each interval of time.

Definition at line 73 of file output.c.

4.3 include/parser.h File Reference

parser.c function declaration.

This graph shows which files directly or indirectly include this file:



Functions

- void parser_run (MPI_Datatype mpi_position_for_possession_type, MPI_Datatype mpi_output_envelope, int possession_processes, picoseconds INTERVAL, char ∗fullgame_path, char ∗interr_path_one, char ∗interr↩ _path_two)

    It receives events from all sensors and communicates with output and possession processes.

4.3.1 Detailed Description

parser.c function declaration.

4.3.2 Function Documentation

4.3.2.1 parser_run()

```
void parser_run (
            MPI_Datatype mpi_position_for_possession_type,
            MPI_Datatype mpi_output_envelope,
            int possession_processes,
            picoseconds INTERVAL,
            char * fullgame_path,
            char * interr_path_one,
            char * interr_path_two )
```

It receives events from all sensors and communicates with output and possession processes.

Start, end and interruptions of the game are highlighted in the data received by the process.

=1mm

spread 0pt [l]|X[-1,l]|X[-1,l]Parameters

Parameters

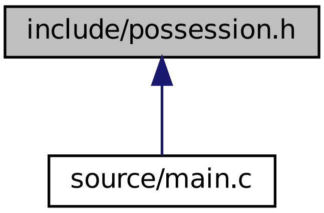
mpi_position_for_possession_type
mpi_output_envelope
possession_processes
INTERVAL

Definition at line 115 of file parser.c.

4.4 include/possession.h File Reference

possession.c function declaration.

This graph shows which files directly or indirectly include this file:



Functions

- double squareDistanceFromBall (position player_position, position ball_last_position)

  This method computes the euclidean distance^2 between a specific player and the ball.
- void possession_run (MPI_Datatype mpi_possession_envelope, MPI_Datatype mpi_output_envelope, unsigned long K)

  Compute ball possessions until the end of the game.

4.4.1 Detailed Description

possession.c function declaration.

4.4.2 Function Documentation

4.4.2.1 possession_run()

```
void possession_run (
             MPI_Datatype mpi_possession_envelope,
             MPI_Datatype mpi_output_envelope,
             unsigned long K )
```

Compute ball possessions until the end of the game.

It keeps waiting for POSITIONS_MESSAGE containing players or ball position updates, until receiving the END↩
OFGAME_MESSAGE or an unknown tag message causing the process to abort.

After receiving a POSITIONS_MESSAGE, it recomputes ball possession: a player is considered in possession of the ball when

- He is the player closest to the ball
- He is not farther than K millimeters from the ball. Then it sends an to the output.c process, which will use it to compute and print the game statistics.

After receiving a ENDOFGAME_MESSAGE, it waits for the sending queue to clear out and abort.

=1mm

spread 0pt [l]|X[-1,l]|X[-1,l]|**Parameters**

Parameters

---

mpi_possession_envelope mpi_datatype of received message from parser_run process, with tag POSITIONS_↩
MESSAGE or ENDOFGAME_MESSAGE.

---

mpi_output_envelope mpi_datatype of sent messages to output process.

K Maximum distance between ball and player: if distance between each player and the ball is greater than k then no one has ball possession. K ranging from 1000 to 5000.

Definition at line 27 of file possession.c.

4.4.2.2 squareDistanceFromBall()

```
double squareDistanceFromBall (
            position player_position,
            position ball_last_position )
```

This method computes the euclidean distance^2 between a specific player and the ball.

$$distance^2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

=1mm

spread 0pt [l]|X[-1,l]|X[-1,l]|**Parameters**

Parameters

player_position	Position of the player we are interested in.
ball_last_position	Ball position.

Returns

Distance^2 between player_position and ball_last_position.

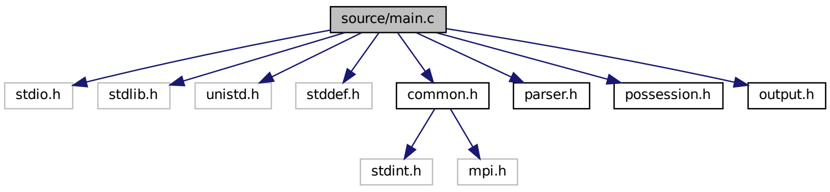
Definition at line 19 of file possession.c.

4.5 source/main.c File Reference

This file contains the main function which starts the program.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stddef.h>
#include "common.h"
#include "parser.h"
#include "possession.h"
#include "output.h"
```

Include dependency graph for main.c:



Functions

- int **main** (int argc, char ∗argv[ ])

4.5.1 Detailed Description

This file contains the main function which starts the program.

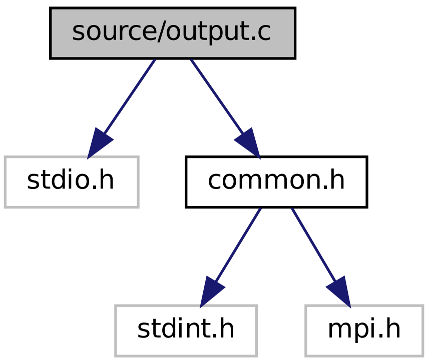
After setting the user-given interval (T in seconds) and possession distance (K in meters), it initializes the M↩
PI execution environment and the MPI datatypes. Given the number of runnable process N, it starts the parser and output processes and N-2 possession process. After all children process have finished it terminates the MPI execution environment and returns.

4.6 source/output.c File Reference

This file defines a process, initialize by main.c, whose job is to compute and output the statistic of the game for each team and player.

```
#include <stdio.h>
#include "common.h"
```

Include dependency graph for output.c:



Functions

- void print_interval (int interval, picoseconds T)

  It prints the interval header with the current game time.
- void print_statistics (const unsigned int ∗interval_possession, const unsigned int ∗total_possession, int interval, picoseconds T)

  It prints for every team and every member last interval statistic, followed by current cumulative statistics.
- void output_run (MPI_Datatype mpi_output_envelope, picoseconds T)

  Core of output's job.

Variables

- const char ∗ **player_names** [ ]
- const picoseconds **FIRST_HALF_DURATION** = FIRST_END - GAME_START
- const picoseconds **SECOND_HALF_DURATION** = GAME_END - SECOND_START

4.6.1 Detailed Description

This file defines a process, initialize by main.c, whose job is to compute and output the statistic of the game for each team and player.

4.6.2 Function Documentation

4.6.2.1 output_run()

```
void output_run (
            MPI_Datatype mpi_output_envelope,
            picoseconds T )
```

Core of output's job.

It keeps waiting for a PRINT_MESSAGE or a POSSESSION_MESSAGE, from onevent or possession processes, until receiving the END_OF_GAME message. After receiving a POSSESSION_MESSAGE, statistics are updated; after receiving a PRINT_MESSAGE, interval and cumulative statistics are printed, by calling print_statistics method, and interval ones are reset; after receiving the END_OF_GAME message, the process exits, after waiting for any pending request. If the received message is of any other type, the process abort.

=1mm

spread 0pt [l]|X[-1,l]|X[-1,l]|**Parameters**

Parameters

mpi_output_envelope	mpi_datatype of received messages.

Definition at line 186 of file output.c.

4.6.2.2 print_interval()

```
void print_interval (
             int interval,
             picoseconds T )
```

It prints the interval header with the current game time.

=1mm

spread 0pt [l]|X[-1,l]|X[-1,l]|**Parameters**

Parameters

interval Current interval id.

T Interval length (in picoseconds).

Definition at line 53 of file output.c.

4.6.2.3 print_statistics()

```
void print_statistics (
             const unsigned int * interval_possession,
             const unsigned int * total_possession,
             int interval,
             picoseconds T )
```

It prints for every team and every member last interval statistic, followed by current cumulative statistics.

=1mm

spread 0pt [l]|X[-1,l]|X[-1,l]|**Parameters**

Parameters

interval_possession Array with last interval statistics for every player (each identified by a constant position in the array).

total_possession Array with cumulative statistics for every player (each identified by a constant position in the array).

interval Incrementing value used to identify each interval of time.

Definition at line 73 of file output.c.

4.6.3 Variable Documentation

4.6.3.1 player_names

```
const char* player_names[]
```

Initial value:

```
= {"None",
                              "Nick Gertje",
                              "Dennis Dotterweich",
                              "Niklas Waelzlein",
                              "Wili Sommer",
                              "Philipp Harlass",
                              "Roman Hartleb",
                              "Erik Engelhardt",
                              "Sandro Schneider",
                              "Leon Krapf",
                              "Kevin Baer",
                              "Luca Ziegler",
                              "Ben Mueller",
                              "Vale Reitstetter",
                              "Christopher Lee",
                              "Leon Heinze",
                              "Leo Langhans"}
```

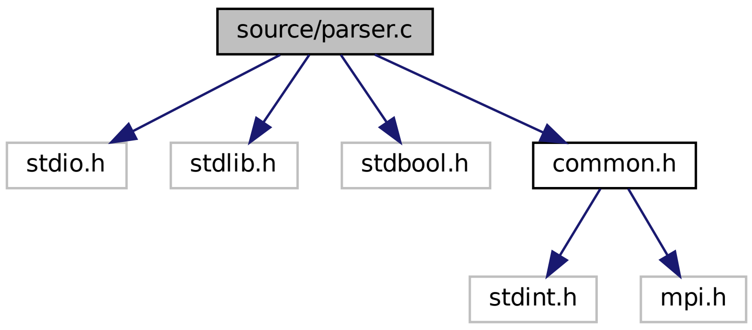
Definition at line 30 of file output.c.

4.7 source/parser.c File Reference

This file defines a process, initialized by main.c, whose job is to read game data.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "common.h"
```

Include dependency graph for parser.c:



Functions

- sensor_type_t **get_sensor_type** (sid_t sid)
- player_t **get_sensor_player** (sid_t sid)
- bool **ball_is_in_play** (position p)
- void **readEvent** (FILE ∗file, event ∗new)
- int **readInterruptionEvent** (FILE ∗∗file, struct interruption_event ∗new, picoseconds start)
- void parser_run (MPI_Datatype mpi_position_for_possession_type, MPI_Datatype mpi_output_envelope, int possession_processes, picoseconds INTERVAL, char ∗fullgame_path, char ∗interr_path_one, char ∗interr←_path_two)

  It receives events from all sensors and communicates with output and possession processes.

Variables

- const sensor_type_t sensor_type_list [ ]
- const player_t sensor_player_list [ ]

4.7.1 Detailed Description

This file defines a process, initialized by main.c, whose job is to read game data.

4.7.2 Function Documentation

4.7.2.1 parser_run()

```
void parser_run (
             MPI_Datatype mpi_position_for_possession_type,
             MPI_Datatype mpi_output_envelope,
             int possession_processes,
             picoseconds INTERVAL,
             char ∗ fullgame_path,
             char ∗ interr_path_one,
             char ∗ interr_path_two )
```

It receives events from all sensors and communicates with output and possession processes.

Start, end and interruptions of the game are highlighted in the data received by the process.

=1mm

spread 0pt [l]|X[-1,l]|X[-1,l]|**Parameters**

Parameters

mpi_position_for_possession_type
mpi_output_envelope
possession_processes
INTERVAL

Definition at line 115 of file parser.c.

4.7.3 Variable Documentation

4.7.3.1 sensor_player_list

```
const player_t sensor_player_list[]
```

Initial value:

```
= {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 2, 0, 0, 4, 0, 0, 0, 6, 6, 0, 0,
                                       0, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 13, 0, 14, 0, 0, 0, 16, 0, 0, 2, 0,
       3, 0, 0, 4,
                                       5, 5, 0, 0, 7, 7, 8, 0, 9, 9, 10, 10, 11, 11, 12, 12, 13, 0, 14, 0,
      15, 15, 16,
                                       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       9, 9}
```

Indexes correspond to sensor ids: for each sensor its player id is stored. Index without an associated player id are stored as 0.

Definition at line 42 of file parser.c.

4.7.3.2 sensor_type_list

```
const sensor_type_t sensor_type_list[]
```

Initial value:

```
= {NONE, NONE, NONE, NONE, BALL, NONE, NONE, NONE, BALL, NONE, BALL, NONE, BALL,
                                          PLAYER, PLAYER, NONE, PLAYER, NONE, NONE, PLAYER, NONE, NONE,
      NONE, PLAYER,
                                          PLAYER, NONE, NONE, NONE, PLAYER, NONE, NONE, NONE, NONE, NONE,
      NONE, NONE,
                                          NONE, NONE, PLAYER, NONE, PLAYER, NONE, NONE, NONE, PLAYER, NONE,
       NONE,
                                          PLAYER, NONE, PLAYER, NONE, NONE, PLAYER, PLAYER, PLAYER, NONE,
      NONE, PLAYER,
                                          PLAYER, PLAYER, NONE, PLAYER, PLAYER, PLAYER, PLAYER, PLAYER,
      PLAYER, PLAYER,
                                          PLAYER, PLAYER, NONE, PLAYER, NONE, PLAYER, PLAYER, PLAYER, NONE,
       NONE, NONE,
                                          NONE, NONE, NONE, NONE, NONE, NONE, NONE, NONE, NONE, PLAYER,
      NONE, NONE,
                                          NONE, NONE, NONE, NONE, NONE, NONE, PLAYER, PLAYER, PLAYER,
      PLAYER, NONE,
                                          NONE, NONE, NONE, REFEREE, REFEREE}
```

Indexes correspond to sensor ids: for each sensor its type is stored. Index without an associated sensor id are stored as NONE.

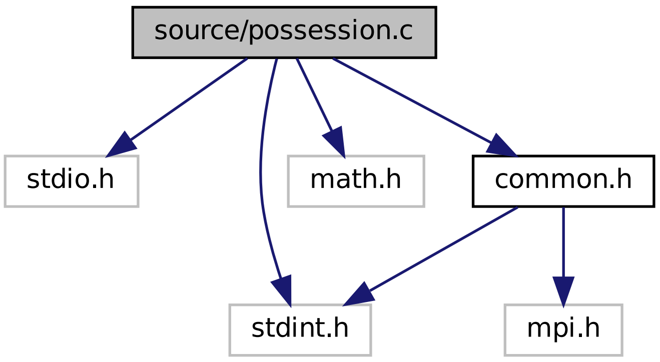
Definition at line 27 of file parser.c.

4.8 source/possession.c File Reference

This file defines a process, initialized by main.c, whose job is to establish which player, and thus team, has the ball, for each game positions update message from the parser_run process.

```
#include <stdio.h>
#include <stdint.h>
#include <math.h>
#include "common.h"
```

Include dependency graph for possession.c:



Functions

- double squareDistanceFromBall (position player_position, position ball_last_position)

  This method computes the euclidean distance^2 between a specific player and the ball.
- void possession_run (MPI_Datatype mpi_possession_envelope, MPI_Datatype mpi_output_envelope, unsigned long K)

  Compute ball possessions until the end of the game.

4.8.1 Detailed Description

This file defines a process, initialized by main.c, whose job is to establish which player, and thus team, has the ball, for each game positions update message from the parser_run process.

4.8.2 Function Documentation

4.8.2.1 possession_run()

```
void possession_run (
            MPI_Datatype mpi_possession_envelope,
            MPI_Datatype mpi_output_envelope,
            unsigned long K )
```

Compute ball possessions until the end of the game.

It keeps waiting for POSITIONS_MESSAGE containing players or ball position updates, until receiving the END↩ OFGAME_MESSAGE or an unknown tag message causing the process to abort.

After receiving a POSITIONS_MESSAGE, it recomputes ball possession: a player is considered in possession of the ball when

- He is the player closest to the ball
- He is not farther than K millimeters from the ball. Then it sends an to the output.c process, which will use it to compute and print the game statistics.

After receiving a ENDOFGAME_MESSAGE, it waits for the sending queue to clear out and abort.

=1mm

spread 0pt [l]|X[-1,l]|X[-1,l]|Parameters

Parameters

mpi_possession_envelope	mpi_datatype of received message from parser_run process, with tag POSITIONS_↩ MESSAGE or ENDOFGAME_MESSAGE.
mpi_output_envelope	mpi_datatype of sent messages to output process.
K	Maximum distance between ball and player: if distance between each player and the ball is greater than k then no one has ball possession. K ranging from 1000 to 5000.

Definition at line 27 of file possession.c.

4.8.2.2 squareDistanceFromBall()

```
double squareDistanceFromBall (
            position player_position,
            position ball_last_position )
```

This method computes the euclidean distance^2 between a specific player and the ball.

$$distance^2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

=1mm

spread 0pt [l]|X[-1,l]|X[-1,l]|**Parameters**

Parameters

player_position	Position of the player we are interested in.
ball_last_position	Ball position.

Returns

Distance^2 between player_position and ball_last_position.

Definition at line 19 of file possession.c.