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

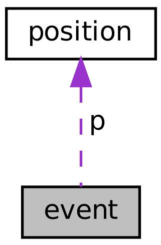
Data Structure Documentation

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 105 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 not updated.

Definition at line 117 of file common.h.

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

- include/common.h

3.3 output_envelope Struct Reference

Used to send messages to the output process.

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

Data Fields

- uint32_t **type**
- uint32_t **content**

3.3.1 Detailed Description

Used to send messages to the output process.

To avoid using multiple messages to know which MPI datatype the next message will be, we use a generic type that works for all the messages that we want to send, i.e. the print message from the parser and the result message from possession. Message types can be POSITIONS_MESSAGE, PRINT_MESSAGE, POSSESSION_MESSAGE and ENDOFGAME_MESSAGE.

Definition at line 145 of file common.h.

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

- include/common.h

3.4 position Struct Reference

Position coordinates in the game field.

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

Data Fields

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

3.4.1 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 91 of file common.h.

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

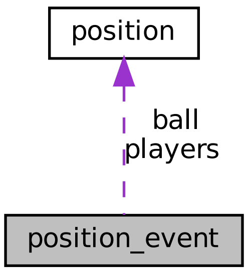
- include/common.h

3.5 position_event Struct Reference

Shows a game snapshot.

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

Collaboration diagram for position_event:



Data Fields

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

3.5.1 Detailed Description

Shows 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 130 of file common.h.

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

- include/common.h

Chapter 4

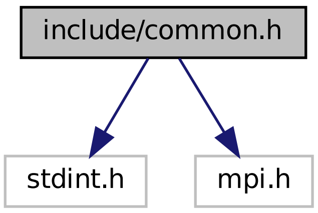
File Documentation

4.1 include/common.h File Reference

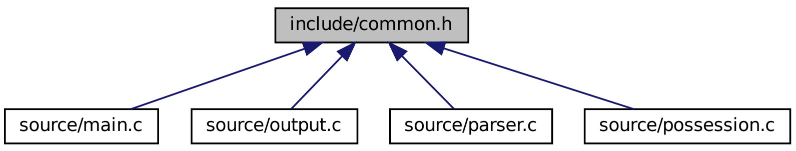
Common constants and definitions.

```
#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

  Shows a game snapshot.
- struct output_envelope

  Used to send messages to the output process.

Macros

- #define PRGDEBUG 0

  Enables additional debug output.
- #define **DBG**(x) /∗nothing∗/
- #define FULLGAME_PATH "../datasets/full-game"

  Default path for the position events file.
- #define SECTOPIC 1000000000000

  Conversion factor from seconds to picosends.
- #define POSSESSION_BUFFER_SIZE 1

  Dimension of the results buffer to increase pipelining.
- #define IGNORE_GOALKEEPER 0

  Set to 1 to remove the goalkeepers from the statistics.

- #define FIRST_INTERRUPTIONS "../datasets/referee-events/Game Interruption/1st Half.csv"

  Default path for the game interruptions file.
- #define SECOND_INTERRUPTIONS "../datasets/referee-events/Game Interruption/2nd Half.csv"

  Default path for the game interruptions file.

- #define XMIN 0

  Field dimensions.
- #define XMAX 52483

  Field dimensions.
- #define YMIN (-33960)

  Field dimensions.
- #define YMAX 33965

  Field dimensions.

- #define GAME_START 10753295594424116

Beginnings and ends of each half of the game.

- #define FIRST_END 12557295594424116

  Beginnings and ends of each half of the game.
- #define SECOND_START 13086639146403495

  Beginnings and ends of each half of the game.
- #define GAME_END 14879639146403495

  Beginnings and ends of each half of the game.

- #define PARSER_RANK 0

  Process identifiers.
- #define OUTPUT_RANK 1

  Process identifiers.
- #define POSSESSION_RANK 2

  Process identifiers.

- #define POSITIONS_MESSAGE 0

  Message type identifiers, defined as integers for use with MPI.
- #define PRINT_MESSAGE 1

  Message type identifiers, defined as integers for use with MPI.
- #define POSSESSION_MESSAGE 2

  Message type identifiers, defined as integers for use with MPI.
- #define ENDOFGAME_MESSAGE 3

  Message type identifiers, defined as integers for use with MPI.

Typedefs

- typedef uint32_t sid_t

  Sensor id type.
- typedef uint32_t player_t

  Player type.
- typedef uint64_t picoseconds

  Picoseconds type.
- 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

  Shows a game snapshot.

Enumerations

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

  Each sensor registers data from a specific PLAYER, from the REFEREE or from the BALL; NONE as default case.

4.1.1 Detailed Description

Common constants and definitions.

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

4.1.2 Typedef Documentation

4.1.2.1 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.

4.1.2.2 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 not updated.

4.1.2.3 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.

4.1.2.4 position_event

```
typedef struct position_event position_event
```

Shows a game snapshot.

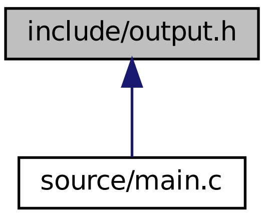
It is characterized by:

- 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)

  Starts the output process.

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 )
```

Starts the output process.

It keeps waiting for a PRINT_MESSAGE or a POSSESSION_MESSAGE, from possession processes, until it receives the END_OF_GAME message. After receiving a POSSESSION_MESSAGE, statistics are updated; after receiving a PRINT_MESSAGE, interval and cumulative statistics are printed, 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.

Parameters

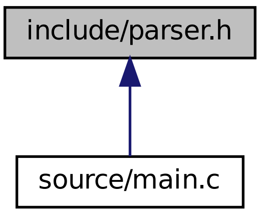
mpi_output_envelope	MPI datatype of the received messages.
T	Length of time between outputs

Definition at line 203 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 T, char ∗fullgame_path, char ∗interr_path_one, char ∗interr_path_two)

  Starts the parser, which receives events from all sensors and communicates with the 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 T,
             char * fullgame_path,
             char * interr_path_one,
             char * interr_path_two )
```

Starts the parser, which receives events from all sensors and communicates with the output and possession processes.

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

Parameters

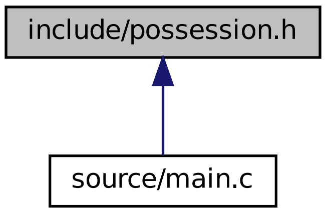
mpi_position_for_possession_type	MPI datatype used to send messages to the possession process
mpi_output_envelope	MPI datatype used to send messages to the output process
possession_processes	Number of possession processes that are running, used to know buffer sizes
T	Length of time between outputs
fullgame_path	Path for the position events file
interr_path_one	Path for the first game interruptions file
interr_path_two	Path for the second game interruptions file

Definition at line 147 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

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

  Starts the possession process, which computes ball possessions given the player positions.

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 )
```

Starts the possession process, which computes ball possessions given the player positions.

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.

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 is in millimeters and ranges from 1000 to 5000.

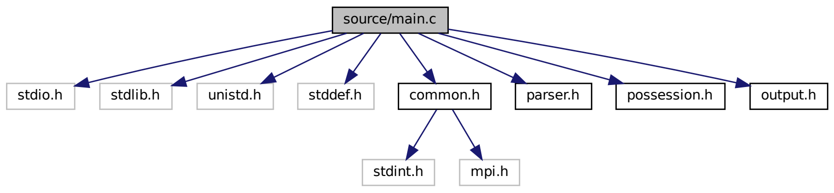
Definition at line 57 of file possession.c.

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.

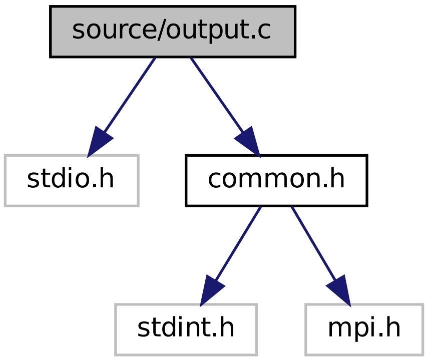
After setting the user-given interval (T in seconds) and possession distance (K in meters), it initializes the M←↩ PI execution environment and the MPI datatypes. Given the number of runnable process N, it starts the parser and output processes and N-2 possession process. After all children process have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

  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)

  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)

  Starts the output process.

Variables

- const char ∗ player_names [ ]

  Names corresponding to each player id.

- const picoseconds FIRST_HALF_DURATION = FIRST_END - GAME_START

  Used to print the interval header.
- const picoseconds SECOND_HALF_DURATION = GAME_END - SECOND_START

  Used to print the interval header.

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 )
```

Starts the output process.

It keeps waiting for a PRINT_MESSAGE or a POSSESSION_MESSAGE, from possession processes, until it receives the END_OF_GAME message. After receiving a POSSESSION_MESSAGE, statistics are updated; after receiving a PRINT_MESSAGE, interval and cumulative statistics are printed, 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.

Parameters

mpi_output_envelope	MPI datatype of the received messages.
T	Length of time between outputs

Definition at line 203 of file output.c.

4.6.2.2 print_interval()

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

Prints the interval header with the current game time.

Parameters

interval	Current interval id.
T	Interval length (in picoseconds).

Definition at line 43 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 )
```

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

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 74 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"}
```

Names corresponding to each player id.

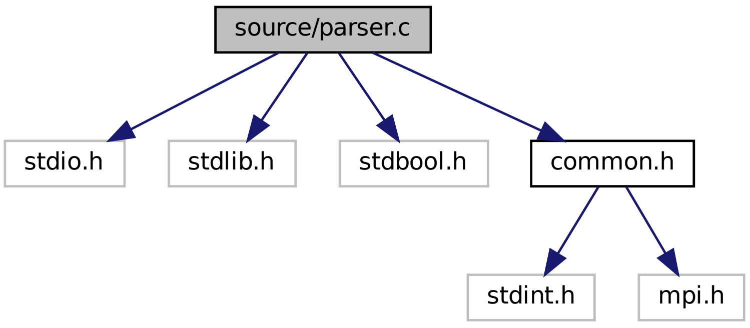
Definition at line 13 of file output.c.

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)

  Given the sensor id it return the sensor type.
- player_t get_sensor_player (sid_t sid)

  Given the sensor id of a player, return the player.
- bool ball_is_in_play (position p)

  Returns TRUE if the given position is within the field.
- void readEvent (FILE ∗file, event ∗new)

  Reads an event from the file and returns it as a new object.
- int readInterruptionEvent (FILE ∗∗file, struct interruption_event ∗new, picoseconds start)

  Reads a new interruption event from file and store read data in the new interruption_event.
- void parser_run (MPI_Datatype mpi_position_for_possession_type, MPI_Datatype mpi_output_envelope, int possession_processes, picoseconds T, char ∗fullgame_path, char ∗interr_path_one, char ∗interr_path_two)

  Starts the parser, which receives events from all sensors and communicates with the 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 ball_is_in_play()

```
bool ball_is_in_play (
             position p )
```

Returns TRUE if the given position is within the field.

Parameters

p	Ball position.

Returns

a bool.

Definition at line 88 of file parser.c.

4.7.2.2 get_sensor_player()

```
player_t get_sensor_player (
             sid_t sid )
```

Given the sensor id of a player, return the player.

Parameters

sid	Sensor id.

Returns

Id of the player as player_t.

Definition at line 70 of file parser.c.

4.7.2.3 get_sensor_type()

```
sensor_type_t get_sensor_type (
             sid_t sid )
```

Given the sensor id it return the sensor type.

Parameters

sid	Sensor id.

Returns

Sensor type; NONE, BALL, PLAYER or REFEREE.

Definition at line 52 of file parser.c.

4.7.2.4 parser_run()

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

Starts the parser, which receives events from all sensors and communicates with the output and possession processes.

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

Parameters

mpi_position_for_possession_type	MPI datatype used to send messages to the possession process
mpi_output_envelope	MPI datatype used to send messages to the output process
possession_processes	Number of possession processes that are running, used to know buffer sizes
T	Length of time between outputs
fullgame_path	Path for the position events file
interr_path_one	Path for the first game interruptions file
interr_path_two	Path for the second game interruptions file

Definition at line 147 of file parser.c.

4.7.2.5 readEvent()

```
void readEvent (
            FILE * file,
            event * new )
```

Reads an event from the file and returns it as a new object.

Parameters

file	An open file pointer to read from
new	A pointer to a free event buffer to write the new event to

Definition at line 97 of file parser.c.

4.7.2.6 readInterruptionEvent()

```
int readInterruptionEvent (
             FILE ** file,
             struct interruption_event * new,
             picoseconds start )
```

Reads a new interruption event from file and store read data in the new interruption_event.

Parameters

file	An open file pointer to read from
new	A pointer to a free interruption_event buffer to write the new event to
start	Start time of the current half of the game. Used as offset for the event time, as the files start from zero

Returns

0 if everything went ok, non-zero if an error occurred

Definition at line 109 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.

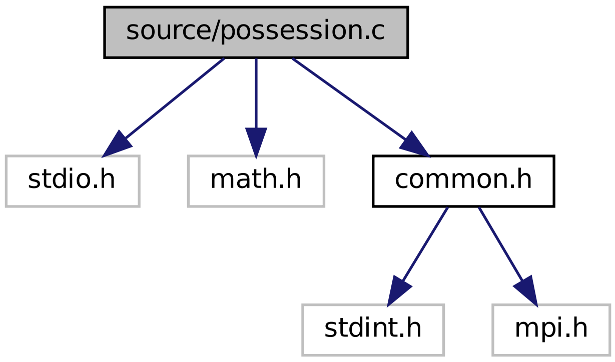
Definition at line 27 of file parser.c.

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 <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)

    Starts the possession process, which computes ball possessions given the player positions.

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 )
```

Starts the possession process, which computes ball possessions given the player positions.

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.

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 is in millimeters and ranges from 1000 to 5000.

Definition at line 57 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.

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

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 26 of file possession.c.

Index