

RoboSoccer Team C

Generated by Doxygen 1.8.6

Thu Jul 28 2016 20:04:10

Contents

1	Namespace Index	1
1.1	Namespace List	1
2	Hierarchical Index	3
2.1	Class Hierarchy	3
3	Class Index	5
3.1	Class List	5
4	File Index	7
4.1	File List	7
5	Namespace Documentation	9
5.1	ATTACKER_STATES Namespace Reference	9
5.1.1	Enumeration Type Documentation	9
5.1.1.1	AttackerStates	9
5.2	CLEARBALL_STATES Namespace Reference	9
5.2.1	Enumeration Type Documentation	9
5.2.1.1	ClearBallStates	9
5.3	DEBUG_STATES Namespace Reference	10
5.3.1	Enumeration Type Documentation	10
5.3.1.1	DebugStates	10
5.4	DEFENDER_STATES Namespace Reference	10
5.4.1	Enumeration Type Documentation	10
5.4.1.1	DefenderStates	10
5.5	GAMECONTROL_STATES Namespace Reference	10
5.5.1	Enumeration Type Documentation	11
5.5.1.1	GameControlStates	11
5.6	GOALKEEPER_KICK_STATES Namespace Reference	11
5.6.1	Enumeration Type Documentation	11
5.6.1.1	GkKickState	11
5.7	GOALKEEPER_STATES Namespace Reference	11
5.7.1	Enumeration Type Documentation	12

5.7.1.1	GoalyState	12
5.8	KICKOFF_STATES Namespace Reference	12
5.8.1	Enumeration Type Documentation	12
5.8.1.1	ClearBallStates	12
5.9	PASSTO_STATES Namespace Reference	12
5.9.1	Enumeration Type Documentation	12
5.9.1.1	PassToStates	12
5.10	SHOOTBALL_STATES Namespace Reference	12
5.10.1	Enumeration Type Documentation	13
5.10.1.1	ShootBallStates	13
5.11	STOPBALL_STATES Namespace Reference	13
5.11.1	Enumeration Type Documentation	13
5.11.1.1	StopBallStates	13
5.12	STRATEGIES Namespace Reference	13
5.12.1	Enumeration Type Documentation	14
5.12.1.1	Strategies	14
5.13	SUPPORT_GK_STATES Namespace Reference	14
5.13.1	Enumeration Type Documentation	14
5.13.1.1	SupportGkStates	14
6	Class Documentation	15
6.1	Agent Class Reference	15
6.1.1	Constructor & Destructor Documentation	18
6.1.1.1	Agent	18
6.1.1.2	~Agent	19
6.1.2	Member Function Documentation	19
6.1.2.1	activate	19
6.1.2.2	activateCA	19
6.1.2.3	addTargetPoint	19
6.1.2.4	addTargetPoint	19
6.1.2.5	addTargetPoints	19
6.1.2.6	addTargetPoints	19
6.1.2.7	avoidPenaltyZone	19
6.1.2.8	calcShootPosition	20
6.1.2.9	cruise	20
6.1.2.10	deactivate	20
6.1.2.11	deactivateCA	20
6.1.2.12	deleteTargetPoints	20
6.1.2.13	getBallTargetLine	20
6.1.2.14	getDriveBackwards	20

6.1.2.15	getPositionFiltered	20
6.1.2.16	isAtTarget	20
6.1.2.17	isBehindBall	20
6.1.2.18	run	21
6.1.2.19	setDesiredSpeed	21
6.1.2.20	setSide	21
6.1.2.21	setTargetPoint	21
6.1.2.22	setTargetPoint	21
6.1.2.23	setTargetPoints	21
6.1.2.24	setTargetPoints	21
6.1.2.25	shoot	21
6.1.2.26	shootBall	21
6.1.2.27	startClearBall	22
6.1.2.28	startPassTo	22
6.1.2.29	startShootBall	22
6.1.2.30	startStopBall	22
6.1.2.31	stopAllActions	22
6.1.2.32	turn	22
6.1.2.33	turn	22
6.1.2.34	turn	22
6.1.2.35	update	22
6.1.3	Friends And Related Function Documentation	22
6.1.3.1	Debug	22
6.1.4	Member Data Documentation	22
6.1.4.1	active_	22
6.1.4.2	attackerModeActive_	22
6.1.4.3	defenderModeActive_	23
6.1.4.4	desiredSpeed_	23
6.1.4.5	driveBackwards	23
6.1.4.6	enemyFieldHalf_	23
6.1.4.7	enemyGoalSegment_	23
6.1.4.8	enemyPenaltyZone_	23
6.1.4.9	heading_	23
6.1.4.10	kd_	23
6.1.4.11	ki_	23
6.1.4.12	kickOffActive_	23
6.1.4.13	kp_	23
6.1.4.14	lastError	23
6.1.4.15	lastPosition_	24
6.1.4.16	lastPositionTimestamp_	24

6.1.4.17	lastSpeed_	24
6.1.4.18	leftSide_	24
6.1.4.19	ourSide_	24
6.1.4.20	ownCornerBottomLeft_	24
6.1.4.21	ownCornerBottomRight_	24
6.1.4.22	ownFieldHalf_	24
6.1.4.23	ownGoalSegment_	24
6.1.4.24	ownPenaltyZone_	24
6.1.4.25	penaltyModeActive_	24
6.1.4.26	rightSide_	24
6.1.4.27	ROBOT_RADIUS	25
6.1.4.28	robotSpeed_	25
6.1.4.29	shootBallActive_	25
6.1.4.30	shootBallSM	25
6.1.4.31	speedIntegrated_	25
6.1.4.32	targetPointMutex_	25
6.1.4.33	timerAgent_	25
6.1.4.34	timerUpdate_	25
6.1.4.35	useCA_	25
6.2	Ball Class Reference	25
6.2.1	Constructor & Destructor Documentation	26
6.2.1.1	Ball	26
6.2.2	Member Function Documentation	26
6.2.2.1	getLastTimestamp	26
6.3	Circle Class Reference	26
6.3.1	Constructor & Destructor Documentation	27
6.3.1.1	Circle	27
6.3.1.2	Circle	27
6.3.1.3	Circle	27
6.3.1.4	Circle	27
6.3.2	Member Function Documentation	27
6.3.2.1	getDistance	27
6.3.2.2	getIntersection	28
6.3.2.3	getIntersection	28
6.3.2.4	getRadius	28
6.3.2.5	getTangentPoints	28
6.3.2.6	getValidPosition	28
6.3.2.7	intersects	28
6.3.2.8	intersects	29
6.3.2.9	isInside	29

6.3.3	Friends And Related Function Documentation	29
6.3.3.1	operator<<	29
6.4	Debug Class Reference	29
6.4.1	Constructor & Destructor Documentation	30
6.4.1.1	Debug	30
6.4.2	Member Function Documentation	30
6.4.2.1	run	30
6.4.3	Friends And Related Function Documentation	30
6.4.3.1	GameControl	30
6.5	Enemy Class Reference	30
6.5.1	Constructor & Destructor Documentation	31
6.5.1.1	Enemy	31
6.5.1.2	~Enemy	31
6.5.2	Member Function Documentation	31
6.5.2.1	getDistToBall	31
6.6	FieldPlayer Class Reference	31
6.6.1	Constructor & Destructor Documentation	32
6.6.1.1	FieldPlayer	32
6.6.1.2	~FieldPlayer	32
6.6.2	Member Function Documentation	32
6.6.2.1	run	32
6.6.2.2	setAttacker	32
6.6.2.3	setDefender	32
6.6.2.4	setKickoffPreparationTarget	33
6.6.2.5	setSide	33
6.6.2.6	startAttackerMode	33
6.6.2.7	startDefenderMode	33
6.6.2.8	startKickOff	33
6.7	GameControl Class Reference	33
6.7.1	Constructor & Destructor Documentation	34
6.7.1.1	GameControl	34
6.7.2	Member Function Documentation	34
6.7.2.1	run	34
6.7.2.2	setFieldPlayer1	34
6.7.2.3	setFieldPlayer2	34
6.7.2.4	setGoalKeeper	34
6.7.2.5	setPermanentStrategy	35
6.7.3	Friends And Related Function Documentation	35
6.7.3.1	Debug	35
6.8	Physics::GameField Struct Reference	35

6.8.1	Constructor & Destructor Documentation	36
6.8.1.1	GameField	36
6.8.2	Member Data Documentation	36
6.8.2.1	CornerBottomLeft	36
6.8.2.2	CornerBottomRight	36
6.8.2.3	CornerTopLeft	36
6.8.2.4	CornerTopRight	36
6.8.2.5	Field	36
6.8.2.6	GoalLeft	36
6.8.2.7	GoalRight	36
6.8.2.8	HalfwayLine	36
6.8.2.9	InnerGoalLeft	36
6.8.2.10	InnerGoalRight	36
6.8.2.11	LeftHalf	36
6.8.2.12	LowerHalf	36
6.8.2.13	ObstacleCornerBottomLeft	36
6.8.2.14	ObstacleCornerBottomRight	36
6.8.2.15	ObstacleCornerTopLeft	36
6.8.2.16	ObstacleCornerTopRight	36
6.8.2.17	ObstacleField	36
6.8.2.18	ObstaclePenaltyAreaLeft	36
6.8.2.19	ObstaclePenaltyAreaRight	37
6.8.2.20	PenaltyAreaLeft	37
6.8.2.21	PenaltyAreaRight	37
6.8.2.22	RightHalf	37
6.8.2.23	UpperHalf	37
6.9	GoalKeeper Class Reference	37
6.9.1	Constructor & Destructor Documentation	38
6.9.1.1	GoalKeeper	38
6.9.1.2	~GoalKeeper	38
6.9.2	Member Function Documentation	38
6.9.2.1	run	38
6.9.2.2	setAttacker	38
6.9.2.3	setFirstDefender	38
6.9.2.4	setSecondDefender	38
6.9.2.5	setSide	38
6.9.2.6	startGoalKeeper	38
6.9.2.7	startPenaltyMode	39
6.9.2.8	stopGoalKeeper	39
6.10	KdTree Class Reference	39

6.10.1	Constructor & Destructor Documentation	39
6.10.1.1	KdTree	39
6.10.1.2	~KdTree	40
6.10.2	Member Function Documentation	40
6.10.2.1	depth	40
6.10.2.2	getChildren	40
6.10.2.3	inObstacle	40
6.10.2.4	insert	40
6.10.2.5	nearest	40
6.10.2.6	nodeCount	40
6.10.2.7	position	40
6.10.2.8	previous	41
6.10.2.9	root	41
6.11	Line Class Reference	41
6.11.1	Constructor & Destructor Documentation	42
6.11.1.1	Line	42
6.11.1.2	Line	42
6.11.1.3	Line	43
6.11.1.4	Line	44
6.11.1.5	Line	44
6.11.1.6	Line	44
6.11.1.7	Line	44
6.11.2	Member Function Documentation	44
6.11.2.1	getAngle	44
6.11.2.2	getClosestPoint	44
6.11.2.3	getClosestPoint	44
6.11.2.4	getDirectionVector	45
6.11.2.5	getDistance	45
6.11.2.6	getDistance	45
6.11.2.7	getIntersection	45
6.11.2.8	getNormalVector	45
6.11.2.9	getReflection	45
6.11.2.10	getSupportVector	45
6.11.2.11	intersects	45
6.11.2.12	isLeftOfLine	45
6.11.2.13	isLeftOfLine	45
6.11.2.14	setDirection	45
6.11.2.15	setDirectionVector	46
6.11.2.16	setSupport	47
6.11.2.17	setSupportVector	47

6.11.3	Friends And Related Function Documentation	47
6.11.3.1	operator<<	47
6.11.4	Member Data Documentation	47
6.11.4.1	directionVector_	47
6.11.4.2	normalVector_	47
6.11.4.3	supportVector_	47
6.12	LineSegment Class Reference	47
6.12.1	Constructor & Destructor Documentation	49
6.12.1.1	LineSegment	49
6.12.1.2	LineSegment	49
6.12.1.3	LineSegment	49
6.12.1.4	LineSegment	49
6.12.1.5	LineSegment	49
6.12.1.6	LineSegment	49
6.12.2	Member Function Documentation	49
6.12.2.1	getClosestPoint	49
6.12.2.2	getClosestPoint	50
6.12.2.3	getDistanceExtended	50
6.12.2.4	getDistanceExtended	50
6.12.2.5	getEndPoint	50
6.12.2.6	getEndVector	50
6.12.2.7	getIntersection	50
6.12.2.8	getIntersection	50
6.12.2.9	getLength	50
6.12.2.10	getMiddlePoint	50
6.12.2.11	getMiddleVector	50
6.12.2.12	getStartPoint	50
6.12.2.13	getStartVector	50
6.12.2.14	intersects	51
6.12.2.15	intersects	51
6.12.2.16	isInSegmentArea	51
6.12.3	Friends And Related Function Documentation	51
6.12.3.1	operator<<	51
6.13	Node Class Reference	51
6.13.1	Constructor & Destructor Documentation	52
6.13.1.1	Node	52
6.13.1.2	~Node	53
6.13.2	Member Function Documentation	53
6.13.2.1	axis	53
6.13.2.2	child	53

6.13.2.3	depth	53
6.13.2.4	farthestChild	53
6.13.2.5	getChildren	53
6.13.2.6	inObstacle	53
6.13.2.7	nearestChild	54
6.13.2.8	nearestChildPointer	55
6.13.2.9	parent	55
6.13.2.10	position	55
6.13.2.11	previous	55
6.14	Obstacle Class Reference	55
6.14.1	Constructor & Destructor Documentation	56
6.14.1.1	Obstacle	56
6.14.1.2	Obstacle	56
6.14.1.3	Obstacle	56
6.14.1.4	Obstacle	57
6.14.1.5	Obstacle	57
6.14.2	Member Function Documentation	57
6.14.2.1	getCenter	57
6.14.2.2	getDistance	57
6.14.2.3	getIntersection	57
6.14.2.4	getIntersection	57
6.14.2.5	getName	58
6.14.2.6	getTangentPoints	58
6.14.2.7	getValidPosition	58
6.14.2.8	intersects	58
6.14.2.9	intersects	58
6.14.2.10	isInside	58
6.14.2.11	setCenter	59
6.14.2.12	setCenter	59
6.14.2.13	setName	59
6.14.3	Friends And Related Function Documentation	59
6.14.3.1	operator<<	59
6.14.4	Member Data Documentation	59
6.14.4.1	center_	59
6.14.4.2	name_	59
6.15	Path Class Reference	59
6.15.1	Constructor & Destructor Documentation	60
6.15.1.1	Path	60
6.15.1.2	~Path	60
6.15.2	Member Function Documentation	60

6.15.2.1	compute	60
6.15.2.2	getId	61
6.15.2.3	initializePath	61
6.15.3	Member Data Documentation	61
6.15.3.1	cachedCAtargetPoints_	61
6.15.3.2	CAtargetPoints_	61
6.15.3.3	id_	61
6.15.3.4	obstacles_	61
6.15.3.5	physics_	61
6.15.3.6	position_	61
6.15.3.7	targetPoints_	61
6.15.3.8	useGameField_	61
6.16	Physics Class Reference	62
6.16.1	Constructor & Destructor Documentation	65
6.16.1.1	Physics	65
6.16.1.2	Physics	65
6.16.2	Member Function Documentation	65
6.16.2.1	addAgent	65
6.16.2.2	addEnemy	66
6.16.2.3	getAgent	66
6.16.2.4	getAgentObstacles	66
6.16.2.5	getBall	66
6.16.2.6	getBallLastPosition	66
6.16.2.7	getBallObstacle	66
6.16.2.8	getBallPositionFiltered	66
6.16.2.9	getBallTrajectory	66
6.16.2.10	getBallVelocity	66
6.16.2.11	getClosestEnemysDistance	67
6.16.2.12	getClosestEnemysDistanceToBall	68
6.16.2.13	getCornerBottomLeft	68
6.16.2.14	getCornerBottomLeftPtr	68
6.16.2.15	getCornerBottomRight	68
6.16.2.16	getCornerBottomRightPtr	68
6.16.2.17	getCornerTopLeft	68
6.16.2.18	getCornerTopLeftPtr	68
6.16.2.19	getCornerTopRight	68
6.16.2.20	getCornerTopRightPtr	68
6.16.2.21	getEnemies	68
6.16.2.22	getEnemy	68
6.16.2.23	getEnemyClosestToBall	69

6.16.2.24	getEnemyObstacles	69
6.16.2.25	getField	69
6.16.2.26	getFieldPtr	69
6.16.2.27	getGoalLeft	69
6.16.2.28	getGoalLeftPtr	69
6.16.2.29	getGoalRight	69
6.16.2.30	getGoalRightPtr	69
6.16.2.31	getHalfwayLine	69
6.16.2.32	getHalfwayLinePtr	69
6.16.2.33	getInnerGoalLeft	69
6.16.2.34	getInnerGoalLeftPtr	69
6.16.2.35	getInnerGoalRight	69
6.16.2.36	getInnerGoalRightPtr	69
6.16.2.37	getLeftHalf	69
6.16.2.38	getLeftHalfPtr	70
6.16.2.39	getLowerHalf	70
6.16.2.40	getLowerHalfPtr	70
6.16.2.41	getNumberOfAgents	70
6.16.2.42	getNumberOfEnemies	70
6.16.2.43	getNumberOfPlayers	70
6.16.2.44	getObstacleCornerBottomLeft	70
6.16.2.45	getObstacleCornerBottomLeftPtr	70
6.16.2.46	getObstacleCornerBottomRight	70
6.16.2.47	getObstacleCornerBottomRightPtr	70
6.16.2.48	getObstacleCornerTopLeft	70
6.16.2.49	getObstacleCornerTopLeftPtr	70
6.16.2.50	getObstacleCornerTopRight	71
6.16.2.51	getObstacleCornerTopRightPtr	71
6.16.2.52	getObstacleField	71
6.16.2.53	getObstacleFieldPtr	71
6.16.2.54	getObstaclePenaltyAreaLeft	71
6.16.2.55	getObstaclePenaltyAreaLeftPtr	71
6.16.2.56	getObstaclePenaltyAreaRight	71
6.16.2.57	getObstaclePenaltyAreaRightPtr	71
6.16.2.58	getPenaltyAreaLeft	71
6.16.2.59	getPenaltyAreaRight	71
6.16.2.60	getPredBallPosition	71
6.16.2.61	getPredBallVelocity	72
6.16.2.62	getRightHalf	73
6.16.2.63	getRightHalfPtr	73

6.16.2.64	getUpperHalf	73
6.16.2.65	getUpperHalfPtr	73
6.16.2.66	initializePhysics	73
6.16.2.67	isInsideGameField	73
6.16.2.68	isInsideLeftHalf	73
6.16.2.69	isInsideLeftPenaltyArea	73
6.16.2.70	isInsideLowerHalf	73
6.16.2.71	isInsidePenaltyArea	74
6.16.2.72	isInsideRightHalf	74
6.16.2.73	isInsideRightPenaltyArea	74
6.16.2.74	isInsideUpperHalf	74
6.16.2.75	run	74
6.16.2.76	startComparePrediction	74
6.16.2.77	stopComparePrediction	74
6.16.3	Member Data Documentation	74
6.16.3.1	BALL_OBSTACLE_RADIUS	75
6.16.3.2	BALL_RADIUS	75
6.16.3.3	CORNER_MARGIN	75
6.16.3.4	FIELD_MARGIN	75
6.16.3.5	PENALTY_AREA_MARGIN	75
6.16.3.6	ROBOT_OBSTACLE_RADIUS	75
6.16.3.7	ROBOT_RADIUS	75
6.17	PointInfo Struct Reference	75
6.17.1	Detailed Description	76
6.17.2	Constructor & Destructor Documentation	76
6.17.2.1	PointInfo	76
6.17.2.2	PointInfo	76
6.17.2.3	PointInfo	76
6.17.3	Member Data Documentation	76
6.17.3.1	Point	76
6.17.3.2	Time	76
6.17.3.3	Velocity	76
6.18	Quadrangle Struct Reference	76
6.18.1	Constructor & Destructor Documentation	77
6.18.1.1	Quadrangle	77
6.18.1.2	Quadrangle	77
6.18.2	Member Function Documentation	77
6.18.2.1	getDistance	77
6.18.2.2	getIntersection	77
6.18.2.3	getIntersection	77

6.18.2.4	getSegments	78
6.18.2.5	getTangentPoints	78
6.18.2.6	getValidPosition	78
6.18.2.7	intersects	78
6.18.2.8	intersects	78
6.18.2.9	isInside	78
6.18.2.10	setCenter	79
6.18.3	Friends And Related Function Documentation	79
6.18.3.1	operator<<	79
6.19	RunnableObject Class Reference	79
6.19.1	Member Enumeration Documentation	80
6.19.1.1	RunStatus	80
6.19.2	Constructor & Destructor Documentation	80
6.19.2.1	RunnableObject	80
6.19.3	Member Function Documentation	80
6.19.3.1	getSleepTime	80
6.19.3.2	restartTimer	80
6.19.3.3	run	80
6.19.3.4	stop	80
6.19.4	Member Data Documentation	80
6.19.4.1	lastSchedule_	80
6.19.4.2	noWarning_	81
6.19.4.3	status_	81
6.19.4.4	timer_	81
6.20	StateMachine Class Reference	81
6.20.1	Constructor & Destructor Documentation	82
6.20.1.1	StateMachine	82
6.20.2	Member Function Documentation	82
6.20.2.1	after	82
6.20.2.2	changeState	82
6.20.2.3	getState	82
6.20.2.4	incTimer	82
6.20.3	Member Data Documentation	82
6.20.3.1	currentState_	82
6.20.3.2	interval_	82
6.20.3.3	lastState_	82
6.20.3.4	stateChangeFlag_	82
6.20.3.5	timer_	83
6.21	TargetPoint Struct Reference	83
6.21.1	Detailed Description	83

6.21.2	Constructor & Destructor Documentation	83
6.21.2.1	TargetPoint	83
6.21.2.2	TargetPoint	84
6.21.2.3	TargetPoint	85
6.21.2.4	TargetPoint	85
6.21.2.5	TargetPoint	85
6.21.2.6	TargetPoint	85
6.21.2.7	TargetPoint	85
6.21.2.8	TargetPoint	86
6.21.2.9	TargetPoint	86
6.21.3	Member Data Documentation	86
6.21.3.1	AnglePrecision	86
6.21.3.2	Brake	86
6.21.3.3	Heading	86
6.21.3.4	Location	86
6.21.3.5	Precision	86
6.22	Trajectory Class Reference	87
6.22.1	Constructor & Destructor Documentation	87
6.22.1.1	Trajectory	87
6.22.2	Member Function Documentation	87
6.22.2.1	getBallTrajectory	87
6.22.2.2	getPredictedBallPosition	87
6.22.2.3	getPredictedBallVelocity	87
6.22.2.4	getPredictedPointInfo	88
6.22.2.5	printTrajectory	88
6.22.2.6	updateTrajectory	88
6.23	Vector2d Class Reference	88
6.23.1	Constructor & Destructor Documentation	89
6.23.1.1	Vector2d	89
6.23.1.2	Vector2d	89
6.23.1.3	Vector2d	90
6.23.1.4	Vector2d	91
6.23.1.5	Vector2d	91
6.23.2	Member Function Documentation	91
6.23.2.1	getAngle	91
6.23.2.2	getAngle	91
6.23.2.3	getDistance	91
6.23.2.4	getDistance	92
6.23.2.5	getLength	93
6.23.2.6	getLengthSquared	93

6.23.2.7	getNormalized	93
6.23.2.8	getTurned	93
6.23.2.9	normalize	93
6.23.2.10	operator*	93
6.23.2.11	operator*	94
6.23.2.12	operator+	94
6.23.2.13	operator-	94
6.23.2.14	operator/	94
6.23.2.15	operator[]	95
6.23.2.16	operator[]	95
6.23.2.17	toPosition	95
6.23.2.18	turn	95
6.23.3	Friends And Related Function Documentation	95
6.23.3.1	operator<<	95
6.23.3.2	operator<<	95
6.23.4	Member Data Documentation	95
6.23.4.1	"@1	95
6.23.4.2	d	95
6.23.4.3	x	95
6.23.4.4	y	95
7	File Documentation	97
7.1	helper/Circle.cpp File Reference	97
7.1.1	Function Documentation	97
7.1.1.1	operator<<	97
7.2	helper/Circle.h File Reference	97
7.3	helper/KdTree.cpp File Reference	97
7.4	helper/KdTree.h File Reference	97
7.5	helper/Line.cpp File Reference	98
7.5.1	Function Documentation	98
7.5.1.1	operator<<	98
7.6	helper/Line.h File Reference	98
7.7	helper/LineSegment.cpp File Reference	98
7.7.1	Function Documentation	98
7.7.1.1	operator<<	98
7.8	helper/LineSegment.h File Reference	98
7.9	helper/Node.cpp File Reference	99
7.10	helper/Node.h File Reference	99
7.11	helper/Obstacle.cpp File Reference	99
7.11.1	Function Documentation	99

7.11.1.1	operator<<	99
7.12	helper/Obstacle.h File Reference	99
7.13	helper/Quadrangle.cpp File Reference	99
7.13.1	Function Documentation	100
7.13.1.1	operator<<	100
7.14	helper/Quadrangle.h File Reference	100
7.15	helper/Trajectory.cpp File Reference	100
7.16	helper/Trajectory.h File Reference	100
7.17	helper/Vector2d.cpp File Reference	100
7.17.1	Function Documentation	101
7.17.1.1	operator-	101
7.17.1.2	operator<<	101
7.17.1.3	operator<<	101
7.18	helper/Vector2d.h File Reference	101
7.19	lib/Agent.h File Reference	101
7.20	lib/Ball.h File Reference	102
7.21	lib/Debug.h File Reference	102
7.22	lib/Enemy.h File Reference	103
7.23	lib/FieldPlayer.h File Reference	103
7.23.1	Enumeration Type Documentation	103
7.23.1.1	DefenderRole	103
7.24	lib/GameControl.h File Reference	104
7.25	lib/Geometry.h File Reference	104
7.26	lib/GoalKeeper.h File Reference	104
7.27	lib/Path.h File Reference	105
7.28	lib/Physics.h File Reference	105
7.28.1	Function Documentation	106
7.28.1.1	constraint	106
7.28.1.2	movingAverage	106
7.29	lib/RunnableObject.h File Reference	106
7.30	lib/StateMachine.h File Reference	106
7.30.1	Macro Definition Documentation	107
7.30.1.1	SM_DURING	107
7.30.1.2	SM_END	107
7.30.1.3	SM_ENTRY	107
7.30.1.4	SM_EXIT	107
7.30.1.5	SUBSM_DURING	108
7.30.1.6	SUBSM_END	108
7.30.1.7	SUBSM_ENTRY	108
7.30.1.8	SUBSM_EXIT	108

7.31	src/Agent.cpp File Reference	108
7.32	src/Debug.cpp File Reference	108
7.33	src/Enemy.cpp File Reference	109
7.34	src/FieldPlayer.cpp File Reference	109
7.35	src/GameControl.cpp File Reference	109
7.36	src/GoalKeeper.cpp File Reference	109
7.37	src/main.cpp File Reference	109
	7.37.1 Function Documentation	109
	7.37.1.1 main	109
7.38	src/Path.cpp File Reference	110
7.39	src/Physics.cpp File Reference	110
	7.39.1 Macro Definition Documentation	110
	7.39.1.1 PHYSICS_UPDATE_INTERVAL	110
Index		111

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

ATTACKER_STATES	9
CLEARBALL_STATES	9
DEBUG_STATES	10
DEFENDER_STATES	10
GAMECONTROL_STATES	10
GOALKEEPER_KICK_STATES	11
GOALKEEPER_STATES	11
KICKOFF_STATES	12
PASSTO_STATES	12
SHOOTBALL_STATES	12
STOPBALL_STATES	13
STRATEGIES	13
SUPPORT_GK_STATES	14

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Physics::GameField	35
KdTree	39
Line	41
LineSegment	47
Node	51
Obstacle	55
Circle	26
Quadrangle	76
Path	59
Agent	15
FieldPlayer	31
GoalKeeper	37
PointInfo	75
RawBall	
Ball	25
RoboControl	
Agent	15
Enemy	30
StateMachine	81
RunnableObject	79
Agent	15
Debug	29
GameControl	33
Physics	62
TargetPoint	83
Trajectory	87
Vector2d	88

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Agent	15
Ball	25
Circle	26
Debug	29
Enemy	30
FieldPlayer	31
GameControl	33
Physics::GameField	35
GoalKeeper	37
KdTree	39
Line	41
LineSegment	47
Node	51
Obstacle	55
Path	59
Physics	62
PointInfo	
A struct storing information about a point on the ball trajectory. Stores Position, Time and Velocity	75
Quadrangle	76
RunnableObject	79
StateMachine	81
TargetPoint	
Defines target point and how it is supposed to be reached	83
Trajectory	87
Vector2d	88

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

helper/Circle.cpp	97
helper/Circle.h	97
helper/KdTree.cpp	97
helper/KdTree.h	97
helper/Line.cpp	98
helper/Line.h	98
helper/LineSegment.cpp	98
helper/LineSegment.h	98
helper/Node.cpp	99
helper/Node.h	99
helper/Obstacle.cpp	99
helper/Obstacle.h	99
helper/Quadrangle.cpp	99
helper/Quadrangle.h	100
helper/Trajectory.cpp	100
helper/Trajectory.h	100
helper/Vector2d.cpp	100
helper/Vector2d.h	101
lib/Agent.h	101
lib/Ball.h	102
lib/Debug.h	102
lib/Enemy.h	103
lib/FieldPlayer.h	103
lib/GameControl.h	104
lib/Geometry.h	104
lib/GoalKeeper.h	104
lib/Path.h	105
lib/Physics.h	105
lib/RunnableObject.h	106
lib/StateMachine.h	106
src/Agent.cpp	108
src/Debug.cpp	108
src/Enemy.cpp	109
src/FieldPlayer.cpp	109
src/GameControl.cpp	109
src/GoalKeeper.cpp	109
src/main.cpp	109
src/Path.cpp	110

src/Physics.cpp	110
---	-----

Chapter 5

Namespace Documentation

5.1 ATTACKER_STATES Namespace Reference

Enumerations

- enum [AttackerStates](#) { [ANTICIPATE](#), [SHOOT](#) }

5.1.1 Enumeration Type Documentation

5.1.1.1 enum ATTACKER_STATES::AttackerStates

Enumerator

ANTICIPATE

SHOOT

5.2 CLEARBALL_STATES Namespace Reference

Enumerations

- enum [ClearBallStates](#) { [INIT](#), [STOP_BALL](#), [NEAR_GOAL](#), [CLEAR](#), [END](#) }

5.2.1 Enumeration Type Documentation

5.2.1.1 enum CLEARBALL_STATES::ClearBallStates

Enumerator

INIT

STOP_BALL

NEAR_GOAL

CLEAR

END

5.3 DEBUG_STATES Namespace Reference

Enumerations

- enum [DebugStates](#) {
 [NO_DEBUG](#), [CRUISE](#), [INTERCEPT](#), [PENALTY](#),
 [START](#), [SHOOT](#), [PREDICTION](#), [REFEREE](#) }

5.3.1 Enumeration Type Documentation

5.3.1.1 enum DEBUG_STATES::DebugStates

Enumerator

NO_DEBUG
CRUISE
INTERCEPT
PENALTY
START
SHOOT
PREDICTION
REFEREE

5.4 DEFENDER_STATES Namespace Reference

Enumerations

- enum [DefenderStates](#) {
 [SUPPORT_GK](#), [CLEAR_BALL](#), [PASS_BALL](#), [SHOOT_ON_GOAL](#),
 [BLOCK_ENEMY](#), [MOVE_ASIDE](#) }

5.4.1 Enumeration Type Documentation

5.4.1.1 enum DEFENDER_STATES::DefenderStates

Enumerator

SUPPORT_GK
CLEAR_BALL
PASS_BALL
SHOOT_ON_GOAL
BLOCK_ENEMY
MOVE_ASIDE

5.5 GAMECONTROL_STATES Namespace Reference

Enumerations

- enum [GameControlStates](#) {
[REFEREE_INIT](#), [BEFORE_KICK_OFF](#), [KICK_OFF](#), [BEFORE_PENALTY](#),
[PENALTY](#), [PLAY_ON](#), [PAUSE](#), [TIME_OVER](#),
[DEBUG_CRUISE](#), [DEBUG_INTERCEPT](#), [DEBUG_SHOOT](#), [DEBUG_PASSTO](#),
[ATTACKER_MODE](#), [DEFENDER_MODE](#) }

5.5.1 Enumeration Type Documentation

5.5.1.1 enum GAMECONTROL_STATES::GameControlStates

Enumerator

REFEREE_INIT
BEFORE_KICK_OFF
KICK_OFF
BEFORE_PENALTY
PENALTY
PLAY_ON
PAUSE
TIME_OVER
DEBUG_CRUISE
DEBUG_INTERCEPT
DEBUG_SHOOT
DEBUG_PASSTO
ATTACKER_MODE
DEFENDER_MODE

5.6 GOALKEEPER_KICK_STATES Namespace Reference

Enumerations

- enum [GkKickState](#) { [PREPARE](#), [SHOOT](#) }

5.6.1 Enumeration Type Documentation

5.6.1.1 enum GOALKEEPER_KICK_STATES::GkKickState

Enumerator

PREPARE
SHOOT

5.7 GOALKEEPER_STATES Namespace Reference

Enumerations

- enum [GoalyState](#) { [AUTO_HOLD_NOT_ACTIVE](#), [AUTO_HOLD_ACTIVE](#), [CLEAR_BALL](#), [PENALTY](#) }

5.7.1 Enumeration Type Documentation

5.7.1.1 enum GOALKEEPER_STATES::GoalyState

Enumerator

AUTO_HOLD_NOT_ACTIVE
AUTO_HOLD_ACTIVE
CLEAR_BALL
PENALTY

5.8 KICKOFF_STATES Namespace Reference

Enumerations

- enum [ClearBallStates](#) { [PREPARE](#), [PREPARE_KICKOFF](#), [SHOOT](#) }

5.8.1 Enumeration Type Documentation

5.8.1.1 enum KICKOFF_STATES::ClearBallStates

Enumerator

PREPARE
PREPARE_KICKOFF
SHOOT

5.9 PASSTO_STATES Namespace Reference

Enumerations

- enum [PassToStates](#) { [INIT](#), [GET_BEHIND_BALL](#), [GET_ON_BALL_TARGET_LINE](#), [PASS_BALL](#), [END](#) }

5.9.1 Enumeration Type Documentation

5.9.1.1 enum PASSTO_STATES::PassToStates

Enumerator

INIT
GET_BEHIND_BALL
GET_ON_BALL_TARGET_LINE
PASS_BALL
END

5.10 SHOOTBALL_STATES Namespace Reference

Enumerations

- enum [ShootBallStates](#) {
INIT, GET_BEHIND_BALL, GET_ON_BALL_GOAL_LINE, SHOOT_BALL,
KICK, END }

5.10.1 Enumeration Type Documentation

5.10.1.1 enum SHOOTBALL_STATES::ShootBallStates

Enumerator

INIT

GET_BEHIND_BALL

GET_ON_BALL_GOAL_LINE

SHOOT_BALL

KICK

END

5.11 STOPBALL_STATES Namespace Reference

Enumerations

- enum [StopBallStates](#) {
INIT, NOT_MOVING_BALL, BEFORE_BALL, OVERTAKE_BALL,
BLOCK_BALL, END }

5.11.1 Enumeration Type Documentation

5.11.1.1 enum STOPBALL_STATES::StopBallStates

Enumerator

INIT

NOT_MOVING_BALL

BEFORE_BALL

OVERTAKE_BALL

BLOCK_BALL

END

5.12 STRATEGIES Namespace Reference

Enumerations

- enum [Strategies](#) { OFFENSIVE, DEFENSIVE }

5.12.1 Enumeration Type Documentation

5.12.1.1 enum STRATEGIES::Strategies

Enumerator

OFFENSIVE

DEFENSIVE

5.13 SUPPORT_GK_STATES Namespace Reference

Enumerations

- enum [SupportGkStates](#) { [SHORTEN_ANGLE](#), [BLOCK](#), [MOVE_AWAY](#) }

5.13.1 Enumeration Type Documentation

5.13.1.1 enum SUPPORT_GK_STATES::SupportGkStates

Enumerator

SHORTEN_ANGLE

BLOCK

MOVE_AWAY

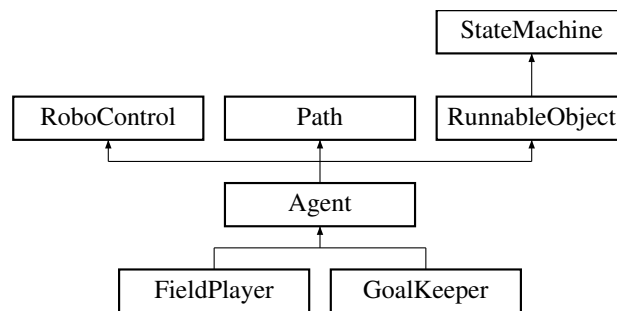
Chapter 6

Class Documentation

6.1 Agent Class Reference

```
#include <Agent.h>
```

Inheritance diagram for Agent:



Public Member Functions

- **Agent** (RTDBConn &DBC, const int deviceNr, **Physics** *physics, int initState, int interval)
constructor for class Agent
- **~Agent** ()
default destructor
- virtual void **run** ()=0
runs state machine
- void **update** ()
updates Timestamp of all agent functions
- void **cruise** ()
moves the robot to its TargetPoint
- void **activateCA** (bool enemies, bool agents, bool ownPenaltyZone, bool enemyPenaltyZone, bool ballObst, bool gameField)
activates Collisionavoidance for different params
- void **deactivateCA** ()
deactivates Collisionavoidance
- bool **turn** (const **TargetPoint** &tp, bool precise=false)
turns the robot to a given Vector or Targetpoint
- bool **turn** (const **Vector2d** &dir, double precision)
- void **turn** (const **Vector2d** &dir, bool precise=false)

- bool [getDriveBackwards](#) ()
- virtual void [setSide](#) (eSide s)=0
set side of team
- void [setTargetPoint](#) (const [TargetPoint](#) &tp)
sets a [TargetPoint](#)
- void [setTargetPoint](#) (const Position &tp)
sets a [TargetPoint](#)
- void [setTargetPoints](#) (const std::vector< [TargetPoint](#) > &tp)
set [TargetPoints](#)
- void [setTargetPoints](#) (const std::vector< Position > &tp)
set [TargetPoints](#)
- void [addTargetPoint](#) (const [TargetPoint](#) &tp)
adds a [TargetPoint](#)
- void [addTargetPoint](#) (const Position &tp)
adds a [TargetPoint](#)
- void [addTargetPoints](#) (const std::vector< [TargetPoint](#) > &tp)
adds [TargetPoints](#)
- void [addTargetPoints](#) (const std::vector< Position > &tp)
adds [TargetPoints](#)
- void [deleteTargetPoints](#) ()
delete all [TargetPoints](#)
- bool [isAtTarget](#) () const
To check if robot has reached his [TargetPoint](#).
- void [activate](#) ()
Activates the drive function of the robot.
- void [deactivate](#) ()
Deactivates the drive function of the robot.
- void [startStopBall](#) ()
Robot stops the ball.
- void [startClearBall](#) ()
If the ball is between roboter and our goal: Roboter overtakes the ball with collision avoidance. After that the roboter drives with full speed to the ball position. Otherwise: Roboter drives with fullspeed to ball position. Is cleared when ball direction is in enemy goal direction.
- void [startShootBall](#) ()
Starts shooting the ball in goal direction.
- void [stopAllActions](#) ()
Stop all actions of the robot.
- void [startPassTo](#) ([Agent](#) *agent)
passes the ball to a roboter of our team
- [Line](#) [getBallTargetLine](#) (Position target)
calculate a [Ball](#) to target line
- void [setDesiredSpeed](#) (double speed)
sets a new drive speed
- [Vector2d](#) [getPositionFiltered](#) ()
get the filtered position
- void [shoot](#) ()
Shoot the ball in right direction.

Public Attributes

- const double [ROBOT_RADIUS](#) = 0.047

Protected Member Functions

- [TargetPoint calcShootPosition](#) (Position ball, Position target, double distToBall=0.1, bool brake=false)
calculate shoot position depending on ball, target and distance to ball
- void [shootBall](#) ()
moves behind ball and shoots it at the enemy goal
- bool [isBehindBall](#) (double val=0)
checks if the robot is behind the ball
- void [avoidPenaltyZone](#) ([TargetPoint](#) *target)
changes the targetpoint if it is in the enemypenaltyZone

Protected Attributes

- Timestamp [lastPositionTimestamp_](#)
- QTime [timerAgent_](#)
- QTime [timerUpdate_](#)
- double [kp_](#) = 0.4
- double [ki_](#) = 0.02
- double [kd_](#) = 0.02
- double [lastError](#) = 0
- double [speedIntegrated_](#) = 0
- double [lastSpeed_](#) = 0
- boost::mutex [targetPointMutex_](#)
- [Vector2d](#) [heading_](#)
- [Vector2d](#) [lastPosition_](#)
- double [robotSpeed_](#) = 0
- double [desiredSpeed_](#) = 0.4
- bool [driveBackwards](#)
- bool [active_](#)
- eSide [ourSide_](#)
- bool [useCA_](#) = true
- [StateMachine](#) [shootBallSM](#)
- bool [attackerModeActive_](#) = false
- bool [defenderModeActive_](#) = false
- bool [shootBallActive_](#) = false
- bool [kickOffActive_](#) = false
- bool [penaltyModeActive_](#) = false
- [Quadrangle](#) * [ownCornerBottomLeft_](#)
- [Quadrangle](#) * [ownCornerBottomRight_](#)
- [LineSegment](#) * [ownGoalSegment_](#)
- [Quadrangle](#) * [ownPenaltyZone_](#)
- [Quadrangle](#) * [ownFieldHalf_](#)
- [LineSegment](#) * [enemyGoalSegment_](#)
- [Quadrangle](#) * [enemyPenaltyZone_](#)
- [Quadrangle](#) * [enemyFieldHalf_](#)
- [Quadrangle](#) * [leftSide_](#)
- [Quadrangle](#) * [rightSide_](#)

Friends

- class [Debug](#)

Additional Inherited Members

6.1.1 Constructor & Destructor Documentation

6.1.1.1 `Agent::Agent (RTDBConn & DBC, const int deviceNr, Physics * physics, int initState, int interval)`

constructor for class [Agent](#)

Parameters

<i>DBC</i>	connection to camera
<i>deviceNr</i>	number of robot
<i>physics</i>	pointer to physics class
<i>initState</i>	initial state
<i>interval</i>	calculation interval

6.1.1.2 Agent::~~Agent ()

default destructor

6.1.2 Member Function Documentation

6.1.2.1 void Agent::activate () [inline]

Activates the drive function of the robot.

6.1.2.2 void Agent::activateCA (bool *enemies*, bool *agents*, bool *ownPenaltyZone*, bool *enemyPenaltyZone*, bool *ballObst*, bool *gameField*)

activates Collisionavoidance for different params

Parameters

<i>activates</i>	enemies
<i>activates</i>	agents
<i>activates</i>	ownPenaltyZone
<i>activates</i>	enemyPenaltyZone
<i>activates</i>	ball
<i>activates</i>	gameField

6.1.2.3 void Agent::addTargetPoint (const TargetPoint & *tp*)

adds a [TargetPoint](#)

6.1.2.4 void Agent::addTargetPoint (const Position & *tp*)

adds a [TargetPoint](#)

6.1.2.5 void Agent::addTargetPoints (const std::vector< TargetPoint > & *tp*)

adds TargetPoints

6.1.2.6 void Agent::addTargetPoints (const std::vector< Position > & *tp*)

adds TargetPoints

6.1.2.7 void Agent::avoidPenaltyZone (TargetPoint * *target*) [protected]

changes the targetpoint if it is in the enemypenaltyZone

6.1.2.8 TargetPoint Agent::calcShootPosition (Position *ball*, Position *target*, double *distToBall* = 0.1, bool *brake* = false) [protected]

calculate shoot position depending on ball, target and distance to ball

6.1.2.9 void Agent::cruise ()

moves the robot to its [TargetPoint](#)

6.1.2.10 void Agent::deactivate () [inline]

Deactivates the drive function of the robot.

6.1.2.11 void Agent::deactivateCA () [inline]

deactivates Collisionavoidance

6.1.2.12 void Agent::deleteTargetPoints () [inline]

delete all TargetPoints

6.1.2.13 Line Agent::getBallTargetLine (Position *target*)

calculate a [Ball](#) to target line

Parameters

<i>target</i>	Position to which you want to shoot
---------------	-------------------------------------

6.1.2.14 bool Agent::getDriveBackwards () [inline]

6.1.2.15 Vector2d Agent::getPositionFiltered () [inline]

get the filtered position

Returns

filtered position of the robot

6.1.2.16 bool Agent::isAtTarget () const

To check if robot has reached his [TargetPoint](#).

Returns

bool: true: Robot has reached his final position, false: robot is still driving

6.1.2.17 bool Agent::isBehindBall (double *val* = 0) [protected]

checks if the robot is behind the ball

Parameters

<i>val</i>	Distance he should be behind the ball
------------	---------------------------------------

6.1.2.18 `virtual void Agent::run ()` [pure virtual]

runs state machine

Implements [RunnableObject](#).

Implemented in [GoalKeeper](#), and [FieldPlayer](#).

6.1.2.19 `void Agent::setDesiredSpeed (double speed)` [inline]

sets a new drive speed

Parameters

<i>speed</i>	speed with which the robot should drive
--------------	---

6.1.2.20 `virtual void Agent::setSide (eSide s)` [pure virtual]

set side of team

Implemented in [FieldPlayer](#), and [GoalKeeper](#).

6.1.2.21 `void Agent::setTargetPoint (const TargetPoint & tp)`

sets a [TargetPoint](#)

6.1.2.22 `void Agent::setTargetPoint (const Position & tp)`

sets a [TargetPoint](#)

6.1.2.23 `void Agent::setTargetPoints (const std::vector< TargetPoint > & tp)`

set TargetPoints

6.1.2.24 `void Agent::setTargetPoints (const std::vector< Position > & tp)`

set TargetPoints

6.1.2.25 `void Agent::shoot ()`

Shoot the ball in right direction.

6.1.2.26 `void Agent::shootBall ()` [protected]

moves behind ball and shoots it at the enemy goal

6.1.2.27 void Agent::startClearBall ()

If the ball is between roboter and our goal: Roboter overtakes the ball with collision avoidance. After that the roboter drives with full speed to the ball position. Otherwise: Roboter drives with fullspeed to ball position. Is cleared when ball direction is in enemy goal direction.

6.1.2.28 void Agent::startPassTo (Agent * agent)

passes the ball to a roboter of our team

Parameters

<i>agent</i>	name of agent the ball shall be passed to
--------------	---

6.1.2.29 void Agent::startShootBall ()

Starts shooting the ball in goal direction.

6.1.2.30 void Agent::startStopBall ()

Robot stops the ball.

6.1.2.31 void Agent::stopAllActions ()

Stop all actions of the robot.

6.1.2.32 bool Agent::turn (const TargetPoint & tp, bool precise = false)

turns the robot to a given Vector or Targetpoint

6.1.2.33 bool Agent::turn (const Vector2d & dir, double precision)**6.1.2.34 void Agent::turn (const Vector2d & dir, bool precise = false)****6.1.2.35 void Agent::update ()**

updates Timestamp of all agent functions

6.1.3 Friends And Related Function Documentation**6.1.3.1 friend class Debug [friend]****6.1.4 Member Data Documentation****6.1.4.1 bool Agent::active_ [protected]**

robot cruise is active

6.1.4.2 bool Agent::attackerModeActive_ = false [protected]

status Atacker Mode

6.1.4.3 `bool Agent::defenderModeActive_ = false` [protected]

status Defender Mode

6.1.4.4 `double Agent::desiredSpeed_ = 0.4` [protected]

desired robot speed

6.1.4.5 `bool Agent::driveBackwards` [protected]

robot is driving backwards

6.1.4.6 `Quadrangle* Agent::enemyFieldHalf_` [protected]

Pointer to enemy field half

6.1.4.7 `LineSegment* Agent::enemyGoalSegment_` [protected]

Pointer to enemy goal segment

6.1.4.8 `Quadrangle* Agent::enemyPenaltyZone_` [protected]

Pointer to enemy penalty zone

6.1.4.9 `Vector2d Agent::heading_` [protected]

heading of the robot

6.1.4.10 `double Agent::kd_ = 0.02` [protected]

differential gain

6.1.4.11 `double Agent::ki_ = 0.02` [protected]

integral gain

6.1.4.12 `bool Agent::kickOffActive_ = false` [protected]

status kickOff

6.1.4.13 `double Agent::kp_ = 0.4` [protected]

proportional gain

6.1.4.14 `double Agent::lastError = 0` [protected]

last deviation

6.1.4.15 Vector2d Agent::lastPosition_ [protected]

last robot position

6.1.4.16 Timestamp Agent::lastPositionTimestamp_ [protected]

time stamp of last position measurement

6.1.4.17 double Agent::lastSpeed_ = 0 [protected]

last speed of the robot

6.1.4.18 Quadrangle* Agent::leftSide_ [protected]

Pointer to lower half of the game field

6.1.4.19 eSide Agent::ourSide_ [protected]

our side

6.1.4.20 Quadrangle* Agent::ownCornerBottomLeft_ [protected]

Pointer to own corner bottomleft

6.1.4.21 Quadrangle* Agent::ownCornerBottomRight_ [protected]

pointer to own corner bottomright

6.1.4.22 Quadrangle* Agent::ownFieldHalf_ [protected]

Pointer to own field half

6.1.4.23 LineSegment* Agent::ownGoalSegment_ [protected]

Pointer to own goal segment

6.1.4.24 Quadrangle* Agent::ownPenaltyZone_ [protected]

Pointer to own penalty zone

6.1.4.25 bool Agent::penaltyModeActive_ = false [protected]

status penalty

6.1.4.26 Quadrangle* Agent::rightSide_ [protected]

Pointer to upper half of the game field

6.1.4.27 `const double Agent::ROBOT_RADIUS = 0.047`

robot radius

6.1.4.28 `double Agent::robotSpeed_ = 0` [protected]

current robot speed

6.1.4.29 `bool Agent::shootBallActive_ = false` [protected]

status shootBall

6.1.4.30 `StateMachine Agent::shootBallSM` [protected]

6.1.4.31 `double Agent::speedIntegrated_ = 0` [protected]

integrator counter

6.1.4.32 `boost::mutex Agent::targetPointMutex_` [protected]

Mutex to lock targetPoints vector

6.1.4.33 `QTime Agent::timerAgent_` [protected]

timeer for agent update

6.1.4.34 `QTime Agent::timerUpdate_` [protected]

time update

6.1.4.35 `bool Agent::useCA_ = true` [protected]

use collision avoidance in general

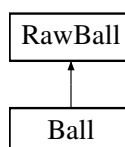
The documentation for this class was generated from the following files:

- [lib/Agent.h](#)
- [src/Agent.cpp](#)

6.2 Ball Class Reference

```
#include <Ball.h>
```

Inheritance diagram for Ball:



Public Member Functions

- [Ball](#) (RTDBConn &DBC)
Constructor for [Ball](#).
- Timestamp [getLastTimestamp](#) ()
return the timestamp of last ball position update

6.2.1 Constructor & Destructor Documentation

6.2.1.1 Ball::Ball (RTDBConn & DBC) [inline]

Constructor for [Ball](#).

6.2.2 Member Function Documentation

6.2.2.1 Timestamp Ball::getLastTimestamp () [inline]

return the timestamp of last ball position update

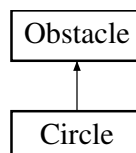
The documentation for this class was generated from the following file:

- lib/[Ball.h](#)

6.3 Circle Class Reference

```
#include <Circle.h>
```

Inheritance diagram for Circle:



Public Member Functions

- [Circle](#) ()
default Constructor of [Circle](#)
- [Circle](#) (const [Circle](#) &circle)
copy Constructor of [Circle](#)
- [Circle](#) (const Position ¢er, double r)
Constructs a circle.
- [Circle](#) (const [Vector2d](#) ¢er, double r)
Constructs a circle.
- double [getRadius](#) () const
Getter for radius of [Circle](#).
- virtual double [getDistance](#) (const Position &pos) const
getter for distance of obstacle to position
- virtual bool [isInside](#) (const Position &pos) const
determines if position is inside an obstacle

- virtual std::vector< [Vector2d](#) > [getIntersection](#) (const [Line](#) &line) const
calculates intersection points between a line and an obstacle
- virtual std::vector< [Vector2d](#) > [getIntersection](#) (const [LineSegment](#) &seg) const
calculates intersection points between a line segment and an obstacle
- virtual bool [intersects](#) (const [Line](#) &line) const
determines if there is an intersection between a line and an obstacle
- virtual bool [intersects](#) (const [LineSegment](#) &seg) const
determines if there is an intersection between a line segment and an obstacle
- virtual std::vector< [Vector2d](#) > [getTangentPoints](#) (const Position &pos) const
calculates tangent points of an obstacle
- virtual Position [getValidPosition](#) (const Position &pos) const
gets the closest valid position (position outside an obstacle)

Friends

- std::ostream & [operator<<](#) (std::ostream &os, const [Circle](#) &vec)

Additional Inherited Members

6.3.1 Constructor & Destructor Documentation

6.3.1.1 [Circle::Circle](#) ()

default Constructor of [Circle](#)

6.3.1.2 [Circle::Circle](#) (const [Circle](#) & circle)

copy Constructor of [Circle](#)

6.3.1.3 [Circle::Circle](#) (const Position & center, double r)

Constructs a circle.

Parameters

<i>center</i>	New center of circle
<i>r</i>	New radius of circle

6.3.1.4 [Circle::Circle](#) (const [Vector2d](#) & center, double r)

Constructs a circle.

Parameters

<i>center</i>	New center of circle
<i>r</i>	New radius of circle

6.3.2 Member Function Documentation

6.3.2.1 double [Circle::getDistance](#) (const Position & p) const [virtual]

getter for distance of obstacle to position

Parameters

<i>p</i>	position
----------	----------

Implements [Obstacle](#).

6.3.2.2 `std::vector< Vector2d > Circle::getIntersection (const Line & line) const` [virtual]

calculates intersection points between a line and an obstacle

Parameters

<i>line</i>	line
-------------	------

Implements [Obstacle](#).

6.3.2.3 `std::vector< Vector2d > Circle::getIntersection (const LineSegment & seg) const` [virtual]

calculates intersection points between a line segment and an obstacle

Parameters

<i>seg</i>	line segment
------------	--------------

Implements [Obstacle](#).

6.3.2.4 `double Circle::getRadius () const` [inline]

Getter for radius of [Circle](#).

6.3.2.5 `std::vector< Vector2d > Circle::getTangentPoints (const Position & pos) const` [virtual]

calculates tangent points of an obstacle

Parameters

<i>pos</i>	point the tangents shall run through
------------	--------------------------------------

Implements [Obstacle](#).

6.3.2.6 `Position Circle::getValidPosition (const Position & pos) const` [virtual]

gets the closest valid position (position outside an obstacle)

Parameters

<i>pos</i>	old position that shall be checked and, if necessary, updated
------------	---

Implements [Obstacle](#).

6.3.2.7 `bool Circle::intersects (const Line & line) const` [virtual]

determines if there is an intersection between a line and an obstacle

Parameters

<i>line</i>	line
-------------	------

Implements [Obstacle](#).

6.3.2.8 `bool Circle::intersects (const LineSegment & seg) const` [virtual]

determines if there is an intersection between a line segment and an obstacle

Parameters

<i>seg</i>	line segment
------------	--------------

Implements [Obstacle](#).

6.3.2.9 `bool Circle::isInside (const Position & p) const` [virtual]

determines if position is inside an obstacle

Parameters

<i>p</i>	position
----------	----------

Implements [Obstacle](#).

6.3.3 Friends And Related Function Documentation

6.3.3.1 `std::ostream& operator<< (std::ostream & os, const Circle & vec)` [friend]

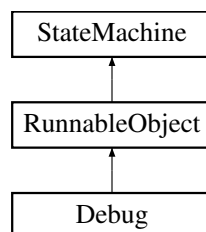
The documentation for this class was generated from the following files:

- [helper/Circle.h](#)
- [helper/Circle.cpp](#)

6.4 Debug Class Reference

```
#include <Debug.h>
```

Inheritance diagram for Debug:



Public Member Functions

- [Debug](#) ([GameControl](#) *gameControl)
constructor for [Debug](#) class
- void [run](#) ()
runs state machine

Friends

- class [GameControl](#)
allows [GameControl](#) to access private fields of [Debug](#) class

Additional Inherited Members

6.4.1 Constructor & Destructor Documentation

6.4.1.1 `Debug::Debug (GameControl * gameControl)`

constructor for [Debug](#) class

Parameters

<i>gameControl</i>	pointer to game control
--------------------	-------------------------

6.4.2 Member Function Documentation

6.4.2.1 `void Debug::run () [virtual]`

runs state machine

Implements [RunnableObject](#).

6.4.3 Friends And Related Function Documentation

6.4.3.1 `friend class GameControl [friend]`

allows [GameControl](#) to access private fields of [Debug](#) class

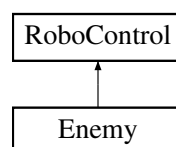
The documentation for this class was generated from the following files:

- [lib/Debug.h](#)
- [src/Debug.cpp](#)

6.5 Enemy Class Reference

```
#include <Enemy.h>
```

Inheritance diagram for Enemy:



Public Member Functions

- [Enemy](#) (RTDBConn &DBC, const int deviceNr, [Physics](#) *physics)
constructor for class [Enemy](#)
- [~Enemy](#) ()

- default destructor*
 - double [getDistToBall](#) ()
calculate distance to ball

6.5.1 Constructor & Destructor Documentation

6.5.1.1 `Enemy::Enemy (RTDBConn & DBC, const int deviceNr, Physics * physics) [inline]`

constructor for class [Enemy](#)

Parameters

<i>DBC</i>	connection to camera
<i>deviceNr</i>	number of robot

6.5.1.2 `Enemy::~Enemy () [inline]`

default destructor

6.5.2 Member Function Documentation

6.5.2.1 `double Enemy::getDistToBall ()`

calculate distance to ball

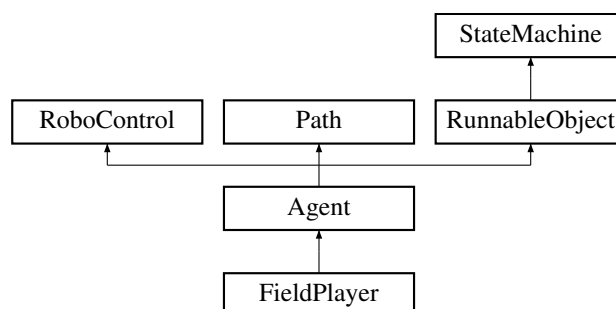
The documentation for this class was generated from the following files:

- lib/[Enemy.h](#)
- src/[Enemy.cpp](#)

6.6 FieldPlayer Class Reference

```
#include <FieldPlayer.h>
```

Inheritance diagram for FieldPlayer:



Public Member Functions

- [FieldPlayer](#) (RTDBConn &DBC, const int deviceNr, [Physics](#) *physics)
constructor for class [FieldPlayer](#)
- [~FieldPlayer](#) ()
default destructor

- void [run](#) ()
runs the state machine
- void [setSide](#) (eSide s)
sets the side of our team
- void [startAttackerMode](#) ()
starts attacker mode
- void [startDefenderMode](#) (DefenderRole role)
starts defender mode
- void [startKickOff](#) ()
starts KickOff
- void [setDefender](#) (FieldPlayer *fp)
set defender, setting attacker to null
- void [setAttacker](#) (FieldPlayer *fp)
set attacker, setting defender to null
- void [setKickoffPreparationTarget](#) (TargetPoint tp)
set a target point for kickoff preparation

Additional Inherited Members

6.6.1 Constructor & Destructor Documentation

6.6.1.1 FieldPlayer::FieldPlayer (RTDBConn & DBC, const int deviceNr, Physics * physics)

constructor for class [FieldPlayer](#)

Parameters

<i>DBC</i>	connection to camera
<i>deviceNr</i>	number of robot
<i>physics</i>	pointer to physics class

6.6.1.2 FieldPlayer::~FieldPlayer ()

default destructor

6.6.2 Member Function Documentation

6.6.2.1 void FieldPlayer::run () [virtual]

runs the state machine

Implements [Agent](#).

6.6.2.2 void FieldPlayer::setAttacker (FieldPlayer * fp) [inline]

set attacker, setting defender to null

6.6.2.3 void FieldPlayer::setDefender (FieldPlayer * fp) [inline]

set defender, setting attacker to null

6.6.2.4 void FieldPlayer::setKickoffPreparationTarget (TargetPoint tp) [inline]

set a target point for kickoff preparation

6.6.2.5 void FieldPlayer::setSide (eSide s) [virtual]

sets the side of our team

Parameters

s	our side
---	----------

Implements [Agent](#).

6.6.2.6 void FieldPlayer::startAttackerMode ()

starts attacker mode

6.6.2.7 void FieldPlayer::startDefenderMode (DefenderRole role)

starts defender mode

6.6.2.8 void FieldPlayer::startKickOff ()

starts KickOff

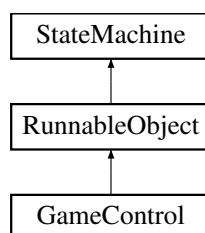
The documentation for this class was generated from the following files:

- lib/[FieldPlayer.h](#)
- src/[FieldPlayer.cpp](#)

6.7 GameController Class Reference

```
#include <GameControl.h>
```

Inheritance diagram for GameController:



Public Member Functions

- [GameControl](#) (Referee *ref, [Physics](#) *physics, eTeam colorT)
constructor
- virtual void [run](#) ()
runs state machine
- void [setGoalKeeper](#) ([GoalKeeper](#) *gk)
sets goalkeeper as class member

- void [setFieldPlayer1](#) ([FieldPlayer](#) *fp1)
sets field player 1 as class member
- void [setFieldPlayer2](#) ([FieldPlayer](#) *fp2)
sets field player 2 as class member
- void [setPermanentStrategy](#) ([STRATEGIES::Strategies](#) strategy)
chooses permanent predefined strategy

Friends

- class [Debug](#)

Additional Inherited Members

6.7.1 Constructor & Destructor Documentation

6.7.1.1 [GameControl::GameControl](#) ([Referee](#) * ref, [Physics](#) * physics, [eTeam](#) colorT)

constructor

Parameters

<i>ref</i>	pointer to referee
<i>physics</i>	pointer to physics
<i>colorT</i>	colour of our team

6.7.2 Member Function Documentation

6.7.2.1 void [GameControl::run](#) () [[virtual](#)]

runs state machine

Implements [RunnableObject](#).

6.7.2.2 void [GameControl::setFieldPlayer1](#) ([FieldPlayer](#) * fp1) [[inline](#)]

sets field player 1 as class member

Parameters

<i>fp1</i>	pointer to field player 1
------------	---------------------------

6.7.2.3 void [GameControl::setFieldPlayer2](#) ([FieldPlayer](#) * fp2) [[inline](#)]

sets field player 2 as class member

Parameters

<i>fp2</i>	pointer to field player 2
------------	---------------------------

6.7.2.4 void [GameControl::setGoalKeeper](#) ([GoalKeeper](#) * gk) [[inline](#)]

sets goalkeeper as class member

Parameters

<i>gk</i>	pointer to goalkeeper
-----------	-----------------------

6.7.2.5 void GameControl::setPermanentStrategy (STRATEGIES::Strategies strategy)

chooses permanent predefined strategy

6.7.3 Friends And Related Function Documentation

6.7.3.1 friend class Debug [friend]

allows [Debug](#) to access private fields of [GameControl](#)

The documentation for this class was generated from the following files:

- lib/[GameControl.h](#)
- src/[GameControl.cpp](#)

6.8 Physics::GameField Struct Reference

```
#include <Physics.h>
```

Public Member Functions

- [GameField](#) ([Quadrangle](#) field, [LineSegment](#) goalLeft, [LineSegment](#) goalRight, [LineSegment](#) innerGoalLeft, [LineSegment](#) innerGoalRight, [LineSegment](#) halfwayLine, [Quadrangle](#) penaltyLeft, [Quadrangle](#) penaltyRight, [Quadrangle](#) obstaclePenaltyLeft, [Quadrangle](#) obstaclePenaltyRight, [Quadrangle](#) obstacleField, [Quadrangle](#) cornerBottomLeft, [Quadrangle](#) cornerBottomRight, [Quadrangle](#) cornerTopRight, [Quadrangle](#) cornerTopLeft, [Quadrangle](#) obstacleCornerBottomLeft, [Quadrangle](#) obstacleCornerBottomRight, [Quadrangle](#) obstacleCornerTopRight, [Quadrangle](#) obstacleCornerTopLeft, [Quadrangle](#) leftHalf, [Quadrangle](#) rightHalf, [Quadrangle](#) upperHalf, [Quadrangle](#) lowerHalf)

Public Attributes

- [Quadrangle](#) Field
- [LineSegment](#) GoalLeft
- [LineSegment](#) GoalRight
- [LineSegment](#) InnerGoalLeft
- [LineSegment](#) InnerGoalRight
- [LineSegment](#) HalfwayLine
- [Quadrangle](#) PenaltyAreaLeft
- [Quadrangle](#) PenaltyAreaRight
- [Quadrangle](#) ObstaclePenaltyAreaLeft
- [Quadrangle](#) ObstaclePenaltyAreaRight
- [Quadrangle](#) ObstacleField
- [Quadrangle](#) CornerBottomLeft
- [Quadrangle](#) CornerBottomRight
- [Quadrangle](#) CornerTopRight
- [Quadrangle](#) CornerTopLeft
- [Quadrangle](#) ObstacleCornerBottomLeft
- [Quadrangle](#) ObstacleCornerBottomRight

- [Quadrangle ObstacleCornerTopRight](#)
- [Quadrangle ObstacleCornerTopLeft](#)
- [Quadrangle LeftHalf](#)
- [Quadrangle RightHalf](#)
- [Quadrangle UpperHalf](#)
- [Quadrangle LowerHalf](#)

6.8.1 Constructor & Destructor Documentation

6.8.1.1 `Physics::GameField::GameField (Quadrangle field, LineSegment goalLeft, LineSegment goalRight, LineSegment innerGoalLeft, LineSegment innerGoalRight, LineSegment halfwayLine, Quadrangle penaltyLeft, Quadrangle penaltyRight, Quadrangle obstaclePenaltyLeft, Quadrangle obstaclePenaltyRight, Quadrangle obstacleField, Quadrangle cornerBottomLeft, Quadrangle cornerBottomRight, Quadrangle cornerTopRight, Quadrangle cornerTopLeft, Quadrangle obstacleCornerBottomLeft, Quadrangle obstacleCornerBottomRight, Quadrangle obstacleCornerTopRight, Quadrangle obstacleCornerTopLeft, Quadrangle leftHalf, Quadrangle rightHalf, Quadrangle upperHalf, Quadrangle lowerHalf) [inline]`

6.8.2 Member Data Documentation

- 6.8.2.1 `Quadrangle Physics::GameField::CornerBottomLeft`
- 6.8.2.2 `Quadrangle Physics::GameField::CornerBottomRight`
- 6.8.2.3 `Quadrangle Physics::GameField::CornerTopLeft`
- 6.8.2.4 `Quadrangle Physics::GameField::CornerTopRight`
- 6.8.2.5 `Quadrangle Physics::GameField::Field`
- 6.8.2.6 `LineSegment Physics::GameField::GoalLeft`
- 6.8.2.7 `LineSegment Physics::GameField::GoalRight`
- 6.8.2.8 `LineSegment Physics::GameField::HalfwayLine`
- 6.8.2.9 `LineSegment Physics::GameField::InnerGoalLeft`
- 6.8.2.10 `LineSegment Physics::GameField::InnerGoalRight`
- 6.8.2.11 `Quadrangle Physics::GameField::LeftHalf`
- 6.8.2.12 `Quadrangle Physics::GameField::LowerHalf`
- 6.8.2.13 `Quadrangle Physics::GameField::ObstacleCornerBottomLeft`
- 6.8.2.14 `Quadrangle Physics::GameField::ObstacleCornerBottomRight`
- 6.8.2.15 `Quadrangle Physics::GameField::ObstacleCornerTopLeft`
- 6.8.2.16 `Quadrangle Physics::GameField::ObstacleCornerTopRight`
- 6.8.2.17 `Quadrangle Physics::GameField::ObstacleField`
- 6.8.2.18 `Quadrangle Physics::GameField::ObstaclePenaltyAreaLeft`

6.8.2.19 **Quadrangle** Physics::GameField::ObstaclePenaltyAreaRight

6.8.2.20 **Quadrangle** Physics::GameField::PenaltyAreaLeft

6.8.2.21 **Quadrangle** Physics::GameField::PenaltyAreaRight

6.8.2.22 **Quadrangle** Physics::GameField::RightHalf

6.8.2.23 **Quadrangle** Physics::GameField::UpperHalf

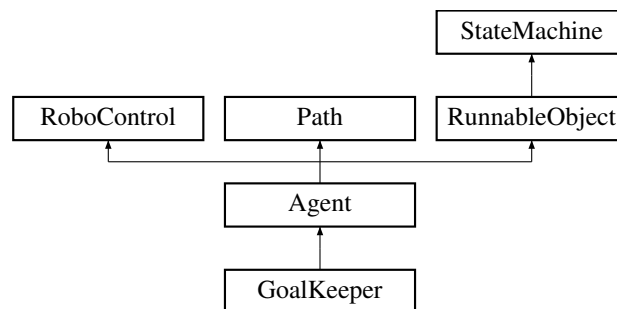
The documentation for this struct was generated from the following file:

- lib/[Physics.h](#)

6.9 GoalKeeper Class Reference

```
#include <GoalKeeper.h>
```

Inheritance diagram for GoalKeeper:



Public Member Functions

- [GoalKeeper](#) (RTDBConn &DBC, const int deviceNr, [Physics](#) *physics)
constructor
- [~GoalKeeper](#) ()
default destructor
- void [setSide](#) (eSide s)
set side of goalkeeper's goal
- void [startPenaltyMode](#) ()
starts penalty mode
- void [startGoalKeeper](#) ()
starts goalkeeper
- void [stopGoalKeeper](#) ()
stops goalkeeper
- void [run](#) ()
runs state machine
- void [setFirstDefender](#) (FieldPlayer *fp)
set the first defender
- void [setSecondDefender](#) (FieldPlayer *fp)
set the second defender, setting attacker to null
- void [setAttacker](#) (FieldPlayer *fp)
set attacker, setting defender to null

Additional Inherited Members

6.9.1 Constructor & Destructor Documentation

6.9.1.1 GoalKeeper::GoalKeeper (RTDBConn & *DBC*, const int *deviceNr*, Physics * *physics*)

constructor

Parameters

<i>DBC</i>	connection to camera
<i>deviceNr</i>	number of roboter
<i>physics</i>	pointer to physics

6.9.1.2 GoalKeeper::~~GoalKeeper ()

default destructor

6.9.2 Member Function Documentation

6.9.2.1 void GoalKeeper::run () [virtual]

runs state machine

Implements [Agent](#).

6.9.2.2 void GoalKeeper::setAttacker (FieldPlayer * *fp*) [inline]

set attacker, setting defender to null

6.9.2.3 void GoalKeeper::setFirstDefender (FieldPlayer * *fp*) [inline]

set the first defender

6.9.2.4 void GoalKeeper::setSecondDefender (FieldPlayer * *fp*) [inline]

set the second defender, setting attacker to null

6.9.2.5 void GoalKeeper::setSide (eSide *s*) [virtual]

set side of goalkeeper's goal

Parameters

<i>s</i>	side of our goal
----------	------------------

Implements [Agent](#).

6.9.2.6 void GoalKeeper::startGoalKeeper ()

starts goalkeeper

6.9.2.7 void GoalKeeper::startPenaltyMode ()

starts penalty mode

6.9.2.8 void GoalKeeper::stopGoalKeeper ()

stops goalkeeper

The documentation for this class was generated from the following files:

- lib/GoalKeeper.h
- src/GoalKeeper.cpp

6.10 KdTree Class Reference

```
#include <KdTree.h>
```

Public Member Functions

- [KdTree](#) (const [Vector2d](#) &position, bool inObstacle)
KdTree constructor.
- [~KdTree](#) ()
KdTree destructor.
- [Node](#) * insert (const [Vector2d](#) &position, bool inObstacle, const [Node](#) *previous)
Inserts a new node into the tree.
- const [Node](#) * nearest (const [Vector2d](#) &position) const
Returns nearest node in tree to position argument.
- unsigned int depth () const
Returns depth of tree.
- unsigned int nodeCount () const
Returns number of nodes in tree.
- const [Node](#) * root () const
Returns root of tree.
- const [Vector2d](#) position (const [Node](#) *node) const
Returns position of node.
- bool inObstacle (const [Node](#) *node) const
Returns inObstacle attribute of node.
- const [Node](#) * previous (const [Node](#) *node) const
Returns predecessor node of node.
- const std::vector< const [Node](#) * > getChildren () const
Returns all children of root.

6.10.1 Constructor & Destructor Documentation

6.10.1.1 KdTree::KdTree (const [Vector2d](#) & position, bool inObstacle)

[KdTree](#) constructor.

Parameters

<i>position</i>	position of root node
<i>inObstacle</i>	root node inObstacle attribute

6.10.1.2 KdTree::~~KdTree ()

KdTree destructor.

6.10.2 Member Function Documentation

6.10.2.1 unsigned int KdTree::depth () const

Returns depth of tree.

6.10.2.2 const std::vector< const Node * > KdTree::getChildren () const

Returns all children of root.

6.10.2.3 bool KdTree::inObstacle (const Node * node) const

Returns inObstacle attribute of node.

Parameters

	node: node to get inObstacle attribute from
--	---

6.10.2.4 Node * KdTree::insert (const Vector2d & position, bool inObstacle, const Node * previous)

Inserts a new node into the tree.

Parameters

<i>position</i>	new node position
<i>inObstacle</i>	new node inObstacle attribute
<i>previous</i>	predecessor of new node

6.10.2.5 const Node * KdTree::nearest (const Vector2d & position) const

Returns nearest node in tree to position argument.

Parameters

<i>position</i>	position to find nearest node from
-----------------	------------------------------------

6.10.2.6 unsigned int KdTree::nodeCount () const [inline]

Returns number of nodes in tree.

6.10.2.7 const Vector2d KdTree::position (const Node * node) const

Returns position of node.

Parameters

	node: node to get position from
--	---------------------------------

6.10.2.8 `const Node * KdTree::previous (const Node * node) const`

Returns predecessor node of node.

Parameters

	node: node to get predecessor
--	-------------------------------

6.10.2.9 `const Node* KdTree::root () const` `[inline]`

Returns root of tree.

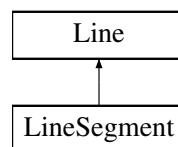
The documentation for this class was generated from the following files:

- [helper/KdTree.h](#)
- [helper/KdTree.cpp](#)

6.11 Line Class Reference

```
#include <Line.h>
```

Inheritance diagram for Line:



Public Member Functions

- [Line](#) ()
default Constructor of [Line](#)
- [Line](#) (const [Line](#) &line)
copy Constructor of [Line](#)
- [Line](#) (Position pos1, Position pos2)
constructing a line using two positions
- [Line](#) (Position pos, double angle)
constructing a line using position and angle
- [Line](#) ([Vector2d](#) vec1, [Vector2d](#) vec2)
constructing a line using two vectors
- [Line](#) ([Vector2d](#) vec, double angle)
constructing a line using vector and angle
- [Line](#) (Position pos, [Vector2d](#) direction)
constructing a line using a position and a direction vector
- double [getDistance](#) (const Position &pos) const
getter for Distance of line to Position
- double [getDistance](#) (const [Vector2d](#) &vec) const

- getter for Distance of line to vector*
- `bool isLeftOfLine (const Vector2d &vec) const`
Checks if position is left of line.
- `bool isLeftOfLine (const Position &pos) const`
Checks if position is left of line.
- `bool intersects (const Line &line) const`
Checks intesection: False if line is parallel, True else.
- `Vector2d getSupportVector () const`
getter for support vector of line
- `Vector2d getDirectionVector () const`
getter for direction vector of line
- `Vector2d getNormalVector () const`
getter for normal vector of line (points left)
- `void setSupportVector (const Vector2d &vec)`
sets support vector of line using a vector
- `void setSupport (const Position &pos)`
sets support vector of line using a position
- `void setDirectionVector (const Vector2d &vec)`
sets direction vector of line using a vector
- `void setDirection (double phi)`
sets direction vector of line using an angle
- `double getAngle (const Line &line) const`
getter for angle of line
- `boost::optional< Vector2d > getIntersection (const Line &line) const`
calculates the Intesection of two lines
- `Vector2d getClosestPoint (const Position &pos) const`
calculates the closest Point on the line to the given Position
- `Vector2d getClosestPoint (const Vector2d &vec) const`
calculates the closest Point on the line to the given Vector
- `Vector2d getReflection (const Line &line) const`
getter for reflection vector of line reflecting at a line segment

Protected Attributes

- `Vector2d supportVector_`
- `Vector2d directionVector_`
- `Vector2d normalVector_`

Friends

- `std::ostream & operator<< (std::ostream &os, const Line &vec)`

6.11.1 Constructor & Destructor Documentation

6.11.1.1 `Line::Line ()`

default Constructor of [Line](#)

6.11.1.2 `Line::Line (const Line & line)`

copy Constructor of [Line](#)

6.11.1.3 Line::Line (Position *pos1*, Position *pos2*)

constructing a line using two positions

Parameters

<i>pos1,2</i>	Position 1,2
---------------	--------------

6.11.1.4 Line::Line (Position *pos*, double *angle*)

constructing a line using position and angle

Parameters

<i>pos</i>	Position
<i>angle</i>	Angle

6.11.1.5 Line::Line (Vector2d *vec1*, Vector2d *vec2*)

constructing a line using two vectors

Parameters

<i>vec1,2</i>	Vector 1,2
---------------	------------

6.11.1.6 Line::Line (Vector2d *vec*, double *angle*)

constructing a line using vector and angle

Parameters

<i>vec</i>	Vector
<i>angle</i>	Angle

6.11.1.7 Line::Line (Position *pos*, Vector2d *direction*)

constructing a line using a position and a direction vector

Parameters

<i>pos</i>	position vector
<i>direction</i>	direction vector

6.11.2 Member Function Documentation

6.11.2.1 double Line::getAngle (const Line & *line*) const [inline]

getter for angle of line

6.11.2.2 Vector2d Line::getClosestPoint (const Position & *pos*) const

calculates the closest Point on the line to the given Position

6.11.2.3 Vector2d Line::getClosestPoint (const Vector2d & *vec*) const

calculates the closest Point on the line to the given Vector

6.11.2.4 Vector2d Line::getDirectionVector () const [inline]

getter for direction vector of line

6.11.2.5 double Line::getDistance (const Position & *pos*) const

getter for Distance of line to Position

6.11.2.6 double Line::getDistance (const Vector2d & *vec*) const

getter for Distance of line to vector

6.11.2.7 boost::optional< Vector2d > Line::getIntersection (const Line & *line*) const

calculates the Intesection of two lines

6.11.2.8 Vector2d Line::getNormalVector () const [inline]

getter for normal vector of line (points left)

6.11.2.9 Vector2d Line::getReflection (const Line & *line*) const

getter for reflection vector of line reflecting at a line segment

6.11.2.10 Vector2d Line::getSupportVector () const [inline]

getter for support vector of line

6.11.2.11 bool Line::intersects (const Line & *line*) const

Checks intesection: False if line is parallel, True else.

6.11.2.12 bool Line::isLeftOfLine (const Vector2d & *vec*) const

Checks if position is left of line.

6.11.2.13 bool Line::isLeftOfLine (const Position & *pos*) const

Checks if position is left of line.

6.11.2.14 void Line::setDirection (double *phi*) [inline]

sets direction vector of line using an angle

Parameters

<i>phi</i>	angle
------------	-------

6.11.2.15 `void Line::setDirectionVector (const Vector2d & vec) [inline]`

sets direction vector of line using a vector

Parameters

<i>vec</i>	Vector
------------	--------

6.11.2.16 void Line::setSupport (const Position & *pos*) [inline]

sets support vector of line using a position

Parameters

<i>pos</i>	Position
------------	----------

6.11.2.17 void Line::setSupportVector (const Vector2d & *vec*) [inline]

sets support vector of line using a vector

Parameters

<i>vec</i>	Vector
------------	--------

6.11.3 Friends And Related Function Documentation

6.11.3.1 std::ostream& operator<< (std::ostream & *os*, const Line & *vec*) [friend]

6.11.4 Member Data Documentation

6.11.4.1 Vector2d Line::directionVector_ [protected]

direction of the line, always normalized

6.11.4.2 Vector2d Line::normalVector_ [protected]

normal vector (orthogonal to directionVector), right-oriented

6.11.4.3 Vector2d Line::supportVector_ [protected]

support vector of the line

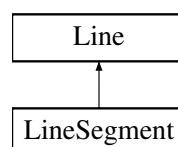
The documentation for this class was generated from the following files:

- [helper/Line.h](#)
- [helper/Line.cpp](#)

6.12 LineSegment Class Reference

```
#include <LineSegment.h>
```

Inheritance diagram for LineSegment:



Public Member Functions

- [LineSegment](#) ()
default constructor of [LineSegment](#)
- [LineSegment](#) (const [LineSegment](#) &seg)
copy constructor of [LineSegment](#)
- [LineSegment](#) (Position pos1, Position pos2)
constructing a line segment using 2 positions
- [LineSegment](#) ([Vector2d](#) vec1, [Vector2d](#) vec2)
constructing a line segment using 2 vectors
- [LineSegment](#) (Position pos, double angle, double length)
constructing a line segment using position, angle and length
- [LineSegment](#) ([Vector2d](#) vec, double angle, double length)
constructing a line segment using vector, angle and length
- double [getDistanceExtended](#) (Position pos) const
calculates distance to line segment
- double [getDistanceExtended](#) (const [Vector2d](#) &vec) const
- bool [isInSegmentArea](#) (const [Vector2d](#) &vec) const
- bool [intersects](#) (const [Line](#) &line) const
- bool [intersects](#) (const [LineSegment](#) &line) const
- Position [getStartPoint](#) () const
getter for starting point of the line segment
- Position [getEndPoint](#) () const
getter for end point of the line segment
- Position [getMiddlePoint](#) () const
getter for middle point of the line segment
- [Vector2d](#) [getStartVector](#) () const
getter for start vector of the line segment
- [Vector2d](#) [getEndVector](#) () const
getter for end vector of the line segment
- [Vector2d](#) [getMiddleVector](#) () const
getter for middle vector of the line segment
- double [getLength](#) () const
getter for the length of the line segment
- boost::optional< [Vector2d](#) > [getIntersection](#) (const [Line](#) &line) const
getter for intersection of line segment and line
- boost::optional< [Vector2d](#) > [getIntersection](#) (const [LineSegment](#) &seg) const
getter for intersection of two line segments
- [Vector2d](#) [getClosestPoint](#) (const Position &pos) const
calculates the closest point on the line segment to the given Position
- [Vector2d](#) [getClosestPoint](#) (const [Vector2d](#) &vec) const
calculates the closest point on the line segment to the given Vector

Friends

- std::ostream & [operator<<](#) (std::ostream &os, const [LineSegment](#) &vec)

Additional Inherited Members

6.12.1 Constructor & Destructor Documentation

6.12.1.1 LineSegment::LineSegment ()

default constructor of [LineSegment](#)

6.12.1.2 LineSegment::LineSegment (const LineSegment & seg)

copy constructor of [LineSegment](#)

6.12.1.3 LineSegment::LineSegment (Position pos1, Position pos2)

constructing a line segment using 2 positions

Parameters

<i>pos1</i>	position 1
<i>pos2</i>	position 2

6.12.1.4 LineSegment::LineSegment (Vector2d vec1, Vector2d vec2)

constructing a line segment using 2 vectors

Parameters

<i>vec1</i>	vector 1,2
-------------	------------

6.12.1.5 LineSegment::LineSegment (Position pos, double angle, double length)

constructing a line segment using position, angle and length

Parameters

<i>pos</i>	position
<i>angle</i>	angle
<i>length</i>	length

6.12.1.6 LineSegment::LineSegment (Vector2d vec, double angle, double length)

constructing a line segment using vector, angle and length

Parameters

<i>vec</i>	vector
<i>angle</i>	angle
<i>length</i>	length

6.12.2 Member Function Documentation

6.12.2.1 Vector2d LineSegment::getClosestPoint (const Position & pos) const

calculates the closest point on the line segment to the given Position

6.12.2.2 Vector2d LineSegment::getClosestPoint (const Vector2d & vec) const

calculates the closest point on the line segment to the given Vector

6.12.2.3 double LineSegment::getDistanceExtended (Position pos) const

calculates distance to line segment

Parameters

<i>pos</i>	position
------------	----------

6.12.2.4 double LineSegment::getDistanceExtended (const Vector2d & vec) const**6.12.2.5 Position LineSegment::getEndPoint () const [inline]**

getter for end point of the line segment

6.12.2.6 Vector2d LineSegment::getEndVector () const [inline]

getter for end vector of the line segment

6.12.2.7 boost::optional< Vector2d > LineSegment::getIntersection (const Line & line) const

getter for intersection of line segment and line

6.12.2.8 boost::optional< Vector2d > LineSegment::getIntersection (const LineSegment & seg) const

getter for intersection of two line segments

6.12.2.9 double LineSegment::getLength () const [inline]

getter for the length of the line segment

6.12.2.10 Position LineSegment::getMiddlePoint () const [inline]

getter for middle point of the line segment

6.12.2.11 Vector2d LineSegment::getMiddleVector () const [inline]

getter for middle vector of the line segment

6.12.2.12 Position LineSegment::getStartPoint () const [inline]

getter for starting point of the line segment

6.12.2.13 Vector2d LineSegment::getStartVector () const [inline]

getter for start vector of the line segment

6.12.2.14 `bool LineSegment::intersects (const Line & line) const`

6.12.2.15 `bool LineSegment::intersects (const LineSegment & line) const`

6.12.2.16 `bool LineSegment::isInSegmentArea (const Vector2d & vec) const`

6.12.3 Friends And Related Function Documentation

6.12.3.1 `std::ostream& operator<< (std::ostream & os, const LineSegment & vec)` `[friend]`

The documentation for this class was generated from the following files:

- [helper/LineSegment.h](#)
- [helper/LineSegment.cpp](#)

6.13 Node Class Reference

```
#include <Node.h>
```

Public Member Functions

- `Node (const Vector2d &position, bool inObstacle, const Node *previous, unsigned int axis, Node *parent)`
Standard constructor for Node.
- `~Node ()`
Deconstructor for Node.
- `Node ** nearestChildPointer (const Vector2d &position)`
Returns pointer to nearest child to position.
- `Node * nearestChild (const Vector2d &position) const`
Return nearest child to position.
- `Node * farthestChild (const Vector2d &position) const`
Returns farthest child to position.
- `const Vector2d & position () const`
Returns position of node.
- `bool inObstacle () const`
Returns inObstacle attribute of node.
- `const Node * previous () const`
Returns predecessor node.
- `unsigned int axis () const`
Returns dividing axis of node.
- `Node * parent () const`
Returns parent node.
- `Node * child (unsigned int index) const`
Returns child defined by index.
- `unsigned int depth () const`
Returns depths of tree starting from this node.
- `void getChildren (std::vector< const Node * > &nodes)`
Returns all children of node.

6.13.1 Constructor & Destructor Documentation

6.13.1.1 `Node::Node (const Vector2d & position, bool inObstacle, const Node * previous, unsigned int axis, Node * parent)`

Standard constructor for [Node](#).

Parameters

<i>position</i>	position of new node
<i>inObstacle</i>	inObstacle attribute of new node
<i>previous</i>	predecessor node of new node
<i>axis</i>	axis that new node divides
<i>parent</i>	parent node of new node

6.13.1.2 Node::~~Node ()

Deconstructor for [Node](#).

6.13.2 Member Function Documentation

6.13.2.1 unsigned int Node::axis () const [inline]

Returns dividing axis of node.

6.13.2.2 Node* Node::child (unsigned int *index*) const [inline]

Returns child defined by index.

Parameters

<i>index</i>	index argument
--------------	----------------

6.13.2.3 unsigned int Node::depth () const

Returns depths of tree starting from this node.

6.13.2.4 Node * Node::farthestChild (const Vector2d & *position*) const

Returns farthest child to position.

Parameters

<i>position</i>	position argument
-----------------	-------------------

6.13.2.5 void Node::getChildren (std::vector< const Node * > & *nodes*)

Returns all children of node.

Parameters

<i>nodes</i>	pointer to vector to save all children
--------------	--

6.13.2.6 bool Node::inObstacle () const [inline]

Returns inObstacle attribute of node.

6.13.2.7 **Node** * Node::nearestChild (const **Vector2d** & *position*) const

Return nearest child to position.

Parameters

<i>position</i>	position argument
-----------------	-------------------

6.13.2.8 Node Node::nearestChildPointer (const Vector2d & position)**

Returns pointer to nearest child to position.

Parameters

<i>position</i>	position argument
-----------------	-------------------

6.13.2.9 Node* Node::parent () const [inline]

Returns parent node.

6.13.2.10 const Vector2d& Node::position () const [inline]

Returns position of node.

6.13.2.11 const Node* Node::previous () const [inline]

Returns predecessor node.

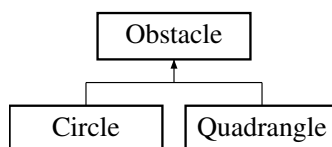
The documentation for this class was generated from the following files:

- [helper/Node.h](#)
- [helper/Node.cpp](#)

6.14 Obstacle Class Reference

```
#include <Obstacle.h>
```

Inheritance diagram for Obstacle:



Public Member Functions

- [Obstacle \(\)](#)
default constructor of Obstacle
- [Obstacle \(const Obstacle &obst\)](#)
copy constructor of Obstacle
- [Obstacle \(const Position &pos\)](#)
constructing an obstacle using its position
- [Obstacle \(string name\)](#)
constructing an obstacle using its name

- [Obstacle](#) (const Position &pos, string name)
constructing an obstacle using its position and its name
- string [getName](#) () const
getter for name of obstacle
- void [setName](#) (string name)
sets name of obstacle
- virtual double [getDistance](#) (const Position &p) const =0
getter for distance of obstacle to position
- virtual bool [isInside](#) (const Position &p) const =0
determines if position is inside an obstacle
- virtual std::vector< [Vector2d](#) > [getIntersection](#) (const [Line](#) &line) const =0
calculates intersection points between a line and an obstacle
- virtual std::vector< [Vector2d](#) > [getIntersection](#) (const [LineSegment](#) &seg) const =0
calculates intersection points between a line segment and an obstacle
- virtual bool [intersects](#) (const [Line](#) &line) const =0
determines if there is an intersection between a line and an obstacle
- virtual bool [intersects](#) (const [LineSegment](#) &seg) const =0
determines if there is an intersection between a line segment and an obstacle
- void [setCenter](#) (const Position &pos)
sets center of an obstacle
- void [setCenter](#) (const [Vector2d](#) &vec)
sets center of an obstacle
- Position [getCenter](#) () const
getter for center of an obstacle
- virtual std::vector< [Vector2d](#) > [getTangentPoints](#) (const Position &pos) const =0
calculates tangent points of an obstacle
- virtual Position [getValidPosition](#) (const Position &pos) const =0
gets the closest valid position (position outside an obstacle)

Protected Attributes

- [Vector2d](#) center_
- string name_

Friends

- std::ostream & [operator<<](#) (std::ostream &os, const [Obstacle](#) &vec)

6.14.1 Constructor & Destructor Documentation

6.14.1.1 [Obstacle::Obstacle](#) () [inline]

default constructor of [Obstacle](#)

6.14.1.2 [Obstacle::Obstacle](#) (const [Obstacle](#) & *obst*) [inline]

copy constructor of [Obstacle](#)

6.14.1.3 [Obstacle::Obstacle](#) (const Position & *pos*) [inline]

constructing an obstacle using its position

Parameters

<i>pos</i>	position as center of obstacle
------------	--------------------------------

6.14.1.4 Obstacle::Obstacle (string *name*) [inline]

constructing an obstacle using its name

Parameters

<i>name</i>	name of obstacle
-------------	------------------

6.14.1.5 Obstacle::Obstacle (const Position & *pos*, string *name*) [inline]

constructing an obstacle using its position and its name

Parameters

<i>pos</i>	position as center of obstacle
<i>name</i>	name of obstacle

6.14.2 Member Function Documentation

6.14.2.1 Position Obstacle::getCenter () const [inline]

getter for center of an obstacle

6.14.2.2 virtual double Obstacle::getDistance (const Position & *p*) const [pure virtual]

getter for distance of obstacle to position

Parameters

<i>p</i>	position
----------	----------

Implemented in [Circle](#), and [Quadrangle](#).

6.14.2.3 virtual std::vector<Vector2d> Obstacle::getIntersection (const Line & *line*) const [pure virtual]

calculates intersection points between a line and an obstacle

Parameters

<i>line</i>	line
-------------	------

Implemented in [Circle](#), and [Quadrangle](#).

6.14.2.4 virtual std::vector<Vector2d> Obstacle::getIntersection (const LineSegment & *seg*) const [pure virtual]

calculates intersection points between a line segment and an obstacle

Parameters

<i>seg</i>	line segment
------------	--------------

Implemented in [Circle](#), and [Quadrangle](#).

6.14.2.5 `string Obstacle::getName () const` `[inline]`

getter for name of obstacle

6.14.2.6 `virtual std::vector<Vector2d> Obstacle::getTangentPoints (const Position & pos) const` `[pure virtual]`

calculates tangent points of an obstacle

Parameters

<i>pos</i>	point the tangents shall run through
------------	--------------------------------------

Implemented in [Circle](#), and [Quadrangle](#).

6.14.2.7 `virtual Position Obstacle::getValidPosition (const Position & pos) const` `[pure virtual]`

gets the closest valid position (position outside an obstacle)

Parameters

<i>pos</i>	old position that shall be checked and, if necessary, updated
------------	---

Implemented in [Circle](#), and [Quadrangle](#).

6.14.2.8 `virtual bool Obstacle::intersects (const Line & line) const` `[pure virtual]`

determines if there is an intersection between a line and an obstacle

Parameters

<i>line</i>	line
-------------	------

Implemented in [Circle](#), and [Quadrangle](#).

6.14.2.9 `virtual bool Obstacle::intersects (const LineSegment & seg) const` `[pure virtual]`

determines if there is an intersection between a line segment and an obstacle

Parameters

<i>seg</i>	line segment
------------	--------------

Implemented in [Circle](#), and [Quadrangle](#).

6.14.2.10 `virtual bool Obstacle::isInside (const Position & p) const` `[pure virtual]`

determines if position is inside an obstacle

Parameters

<i>p</i>	position
----------	----------

Implemented in [Circle](#), and [Quadrangle](#).

6.14.2.11 `void Obstacle::setCenter (const Position & pos)` `[inline]`

sets center of an obstacle

Parameters

<i>pos</i>	new center position
------------	---------------------

6.14.2.12 `void Obstacle::setCenter (const Vector2d & vec)` `[inline]`

sets center of an obstacle

Parameters

<i>vec</i>	new center position
------------	---------------------

6.14.2.13 `void Obstacle::setName (string name)` `[inline]`

sets name of obstacle

6.14.3 Friends And Related Function Documentation

6.14.3.1 `std::ostream& operator<< (std::ostream & os, const Obstacle & vec)` `[friend]`

6.14.4 Member Data Documentation

6.14.4.1 `Vector2d Obstacle::center_` `[protected]`

center of obstacle

6.14.4.2 `string Obstacle::name_` `[protected]`

name of obstacle

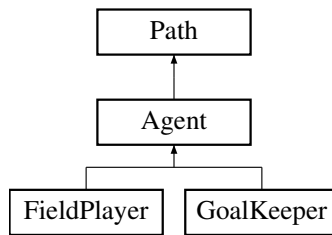
The documentation for this class was generated from the following file:

- [helper/Obstacle.h](#)

6.15 Path Class Reference

```
#include <Path.h>
```

Inheritance diagram for Path:



Public Member Functions

- [Path](#) ([Physics](#) *physics, int id)
constructor of class [Path](#)
- [~Path](#) ()
default destructor
- void [compute](#) ([TargetPoint](#) requestedEnd)
computes path to target point
- void [initializePath](#) ()
initializes path and obstacles
- int [getId](#) ()
Getter for rfcomm id.

Protected Attributes

- [Physics](#) * [physics_](#)
- std::vector< [TargetPoint](#) > [targetPoints_](#)
- std::vector< [TargetPoint](#) > [CAtargetPoints_](#)
- std::vector< [TargetPoint](#) > [cachedCAtargetPoints_](#)
- [Vector2d](#) [position_](#)
- int [id_](#)
- std::vector< [Obstacle](#) * > [obstacles_](#)
- bool [useGameField_](#) = true

6.15.1 Constructor & Destructor Documentation

6.15.1.1 [Path::Path](#) ([Physics](#) * *physics*, int *id*)

constructor of class [Path](#)

Parameters

<i>physics</i>	pointer to physics
<i>id</i>	roboter ID

6.15.1.2 [Path::~Path](#) ()

default destructor

6.15.2 Member Function Documentation

6.15.2.1 void [Path::compute](#) ([TargetPoint](#) *requestedEnd*)

computes path to target point

Parameters

<i>requestedEnd</i>	desired target point
---------------------	----------------------

6.15.2.2 `int Path::getId () [inline]`

Getter for rfcmm id.

6.15.2.3 `void Path::initializePath ()`

initializes path and obstacles

6.15.3 Member Data Documentation

6.15.3.1 `std::vector<TargetPoint> Path::cachedCAtargetPoints_ [protected]`

vector of cached waypoints of collision avoidance trajectory

6.15.3.2 `std::vector<TargetPoint> Path::CAtargetPoints_ [protected]`

vector of waypoints of collision avoidance trajectory

6.15.3.3 `int Path::id_ [protected]`

roboter ID

6.15.3.4 `std::vector<Obstacle*> Path::obstacles_ [protected]`

vector of obstacles

6.15.3.5 `Physics* Path::physics_ [protected]`

pointer to physics

6.15.3.6 `Vector2d Path::position_ [protected]`

position

6.15.3.7 `std::vector<TargetPoint> Path::targetPoints_ [protected]`

vector of target points

6.15.3.8 `bool Path::useGameField_ = true [protected]`

Use Game Field [Obstacle](#) in collision avoidance

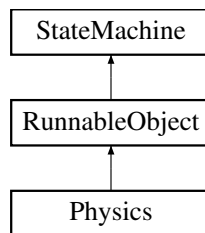
The documentation for this class was generated from the following files:

- [lib/Path.h](#)
- [src/Path.cpp](#)

6.16 Physics Class Reference

```
#include <Physics.h>
```

Inheritance diagram for Physics:



Classes

- struct [GameField](#)

Public Member Functions

- [Physics](#) ()
default constructor
- [Physics](#) ([Ball](#) *ball)
copy constructor
- void [initializePhysics](#) ()
initializes obstacles
- void [addAgent](#) ([Agent](#) *ag)
adds Agent
- void [addEnemy](#) ([Enemy](#) *em)
adds Enemy
- [Enemy](#) * [getEnemy](#) (int nr)
returns enemy specified by argument
- [Agent](#) * [getAgent](#) (int nr)
returns agent specified by argument
- [Ball](#) * [getBall](#) ()
returns ball
- std::vector< [Enemy](#) * > [getEnemies](#) ()
returns all enemies as vector
- int [getNumberOfAgents](#) () const
returns number of agents
- int [getNumberOfEnemies](#) () const
returns number of enemies
- int [getNumberOfPlayers](#) () const
returns number of robots in total
- bool [isInsideGameField](#) ([Position](#) const &pos) const
checks if position is inside field
- bool [isInsideLeftPenaltyArea](#) ([Position](#) const &pos) const
checks if position is inside left penalty area
- bool [isInsideRightPenaltyArea](#) ([Position](#) const &pos) const
checks if position is inside right penalty area
- bool [isInsidePenaltyArea](#) ([Position](#) const &pos) const

- checks if position is inside any penalty area
- bool [isInsideLeftHalf](#) (Position const &pos) const
checks if position is inside left half
- bool [isInsideRightHalf](#) (Position const &pos) const
checks if position is inside right half
- bool [isInsideUpperHalf](#) (Position const &pos) const
checks if position is inside upper half
- bool [isInsideLowerHalf](#) (Position const &pos) const
checks if position is inside lower half
- [LineSegment](#) [getGoalLeft](#) () const
returns left goal line
- [LineSegment](#) [getGoalRight](#) () const
returns right goal line
- [LineSegment](#) [getInnerGoalLeft](#) () const
- [LineSegment](#) [getInnerGoalRight](#) () const
- [LineSegment](#) [getHalfwayLine](#) () const
- [Quadrangle](#) [getPenaltyAreaLeft](#) () const
returns left penalty area
- [Quadrangle](#) [getPenaltyAreaRight](#) () const
returns right penalty area
- [Quadrangle](#) [getObstaclePenaltyAreaLeft](#) () const
returns left penalty area with collision avoidance margin
- [Quadrangle](#) [getObstaclePenaltyAreaRight](#) () const
returns right penalty area with collision avoidance margin
- [Quadrangle](#) [getField](#) () const
returns game field
- [Quadrangle](#) [getObstacleField](#) () const
returns game field with collision avoidance margin
- [Quadrangle](#) [getCornerBottomLeft](#) () const
returns bottom left corner
- [Quadrangle](#) [getCornerBottomRight](#) () const
returns bottom right corner
- [Quadrangle](#) [getCornerTopRight](#) () const
returns top right corner
- [Quadrangle](#) [getCornerTopLeft](#) () const
returns top left corner
- [Quadrangle](#) [getObstacleCornerBottomLeft](#) () const
returns bottom left corner with collision avoidance margin
- [Quadrangle](#) [getObstacleCornerBottomRight](#) () const
returns bottom right corner with collision avoidance margin
- [Quadrangle](#) [getObstacleCornerTopRight](#) () const
returns top right corner with collision avoidance margin
- [Quadrangle](#) [getObstacleCornerTopLeft](#) () const
returns top left corner with collision avoidance margin
- [Quadrangle](#) [getLeftHalf](#) () const
returns left half of field
- [Quadrangle](#) [getRightHalf](#) () const
returns right half of field
- [Quadrangle](#) [getUpperHalf](#) () const
returns upper half of field
- [Quadrangle](#) [getLowerHalf](#) () const

- returns lower half of field*
- `LineSegment * getGoalLeftPtr ()`
returns pointer to left goal line
- `LineSegment * getGoalRightPtr ()`
- `LineSegment * getInnerGoalLeftPtr ()`
- `LineSegment * getInnerGoalRightPtr ()`
- `LineSegment * getHalfwayLinePtr ()`
returns pointer to middle line
- `Quadrangle * getObstaclePenaltyAreaLeftPtr ()`
returns pointer to left penalty area with collision avoidance margin
- `Quadrangle * getObstaclePenaltyAreaRightPtr ()`
returns pointer to right penalty area with collision avoidance margin
- `Quadrangle * getFieldPtr ()`
returns pointer to game field
- `Quadrangle * getObstacleFieldPtr ()`
returns pointer to game field with collision avoidance margin
- `Quadrangle * getCornerBottomLeftPtr ()`
returns pointer to bottom left corner
- `Quadrangle * getCornerBottomRightPtr ()`
returns pointer to bottom right corner
- `Quadrangle * getCornerTopRightPtr ()`
returns pointer to top right corner
- `Quadrangle * getCornerTopLeftPtr ()`
returns pointer to top left corner
- `Quadrangle * getObstacleCornerBottomLeftPtr ()`
returns pointer to bottom left corner with collision avoidance margin
- `Quadrangle * getObstacleCornerBottomRightPtr ()`
returns pointer to bottom right corner with collision avoidance margin
- `Quadrangle * getObstacleCornerTopRightPtr ()`
returns pointer to top right corner with collision avoidance margin
- `Quadrangle * getObstacleCornerTopLeftPtr ()`
returns pointer to top left corner with collision avoidance margin
- `Quadrangle * getLeftHalfPtr ()`
returns pointer to left half of field
- `Quadrangle * getRightHalfPtr ()`
returns pointer to right half of field
- `Quadrangle * getUpperHalfPtr ()`
returns pointer to upper half of field
- `Quadrangle * getLowerHalfPtr ()`
returns pointer to lower half of field
- `Obstacle * getBallObstacle ()`
returns ball with collision avoidance margin
- `std::vector< Obstacle * > getAgentObstacles ()`
returns all agents with collision avoidance margin
- `std::vector< Obstacle * > getEnemyObstacles ()`
returns all enemies with collision avoidance margin
- `Vector2d getBallPositionFiltered ()`
returns filtered ball position
- `Position getBallLastPosition ()`
returns last ball position
- `Position getPredBallPosition (int milliseconds) const`

- returns predicted ball position*
- [Vector2d](#) [getPredBallVelocity](#) (int milliseconds) const
returns predicted ball velocity
- [Vector2d](#) [getBallVelocity](#) () const
returns current ball velocity
- std::vector< [LineSegment](#) > [getBallTrajectory](#) (int milliseconds) const
returns predicted ball trajectory
- [Enemy](#) * [getEnemyClosestToBall](#) ()
returns enemy closest to ball
- double [getClosestEnemysDistance](#) (const [Vector2d](#) &vec) const
returns distance to closest enemy from argument position
- double [getClosestEnemysDistanceToBall](#) () const
returns distance to ball of enemy closest to ball
- void [run](#) ()
main update routine of [Physics](#)
- void [startComparePrediction](#) (int predictionTime)
start a prediction comparison to check prediction calculation
- void [stopComparePrediction](#) ()
stop the prediction comparison

Public Attributes

- const double [ROBOT_RADIUS](#) = 0.047
- const double [ROBOT_OBSACLE_RADIUS](#) = 3.5 * [ROBOT_RADIUS](#)
- const double [BALL_RADIUS](#) = 0.021335
- const double [BALL_OBSACLE_RADIUS](#) = [BALL_RADIUS](#) + 1.5 * [ROBOT_RADIUS](#)
- const double [PENALTY_AREA_MARGIN](#) = 1.05 * [ROBOT_RADIUS](#)
- const double [FIELD_MARGIN](#) = 1.2 * [ROBOT_RADIUS](#)
- const double [CORNER_MARGIN](#) = 1.1 * [ROBOT_RADIUS](#)

Additional Inherited Members

6.16.1 Constructor & Destructor Documentation

6.16.1.1 [Physics::Physics](#) ()

default contructor

6.16.1.2 [Physics::Physics](#) ([Ball](#) * *ball*)

copy constructor

Parameters

<i>ball</i>	initializes member ball_
-------------	--------------------------

6.16.2 Member Function Documentation

6.16.2.1 void [Physics::addAgent](#) ([Agent](#) * *ag*)

adds [Agent](#)

Parameters

<i>ag</i>	Agent to be added
-----------	-----------------------------------

6.16.2.2 void Physics::addEnemy ([Enemy](#) * *em*)

adds [Enemy](#)

Parameters

<i>em</i>	Enemy to be added
-----------	-----------------------------------

6.16.2.3 [Agent](#)* Physics::getAgent (int *nr*) [inline]

returns agent specified by argument

Parameters

<i>nr</i>	id of agent
-----------	-------------

6.16.2.4 std::vector<[Obstacle](#)> Physics::getAgentObstacles () [inline]

returns all agents with collision avoidance margin

6.16.2.5 [Ball](#)* Physics::getBall () [inline]

returns ball

6.16.2.6 Position Physics::getBallLastPosition () [inline]

returns last ball position

6.16.2.7 [Obstacle](#)* Physics::getBallObstacle () [inline]

returns ball with collision avoidance margin

6.16.2.8 [Vector2d](#) Physics::getBallPositionFiltered () [inline]

returns filtered ball position

6.16.2.9 std::vector< [LineSegment](#) > Physics::getBallTrajectory (int *milliseconds*) const

returns predicted ball trajectory

Parameters

<i>milliseconds</i>	prediction time
---------------------	-----------------

6.16.2.10 [Vector2d](#) Physics::getBallVelocity () const

returns current ball velocity

6.16.2.11 `double Physics::getClosestEnemysDistance (const Vector2d & vec) const`

returns distance to closest enemy from argument position

Parameters

<code>vec</code>	position to be checked
------------------	------------------------

6.16.2.12 `double Physics::getClosestEnemysDistanceToBall () const`

returns distance to ball of enemy closest to ball

6.16.2.13 `Quadrangle Physics::getCornerBottomLeft () const` `[inline]`

returns bottom left corner

6.16.2.14 `Quadrangle* Physics::getCornerBottomLeftPtr ()` `[inline]`

returns pointer to bottom left corner

6.16.2.15 `Quadrangle Physics::getCornerBottomRight () const` `[inline]`

returns bottom right corner

6.16.2.16 `Quadrangle* Physics::getCornerBottomRightPtr ()` `[inline]`

returns pointer to bottom right corner

6.16.2.17 `Quadrangle Physics::getCornerTopLeft () const` `[inline]`

returns top left corner

6.16.2.18 `Quadrangle* Physics::getCornerTopLeftPtr ()` `[inline]`

returns pointer to top left corner

6.16.2.19 `Quadrangle Physics::getCornerTopRight () const` `[inline]`

returns top right corner

6.16.2.20 `Quadrangle* Physics::getCornerTopRightPtr ()` `[inline]`

returns pointer to top right corner

6.16.2.21 `std::vector<Enemy*> Physics::getEnemies ()` `[inline]`

returns all enemies as vector

6.16.2.22 `Enemy* Physics::getEnemy (int nr)` `[inline]`

returns enemy specified by argument

Parameters

<i>nr</i>	id of enemy
-----------	-------------

6.16.2.23 Enemy * Physics::getEnemyClosestToBall ()

returns enemy closest to ball

6.16.2.24 std::vector<Obstacle*> Physics::getEnemyObstacles () [inline]

returns all enemies with collision avoidance margin

6.16.2.25 Quadrangle Physics::getField () const [inline]

returns game field

6.16.2.26 Quadrangle* Physics::getFieldPtr () [inline]

returns pointer to game field

6.16.2.27 LineSegment Physics::getGoalLeft () const [inline]

returns left goal line

6.16.2.28 LineSegment* Physics::getGoalLeftPtr () [inline]

returns pointer to left goal line

6.16.2.29 LineSegment Physics::getGoalRight () const [inline]

returns right goal line

6.16.2.30 LineSegment* Physics::getGoalRightPtr () [inline]**6.16.2.31 LineSegment Physics::getHalfwayLine () const [inline]****6.16.2.32 LineSegment* Physics::getHalfwayLinePtr () [inline]**

returns pointer to middle line

6.16.2.33 LineSegment Physics::getInnerGoalLeft () const [inline]**6.16.2.34 LineSegment* Physics::getInnerGoalLeftPtr () [inline]****6.16.2.35 LineSegment Physics::getInnerGoalRight () const [inline]****6.16.2.36 LineSegment* Physics::getInnerGoalRightPtr () [inline]****6.16.2.37 Quadrangle Physics::getLeftHalf () const [inline]**

returns left half of field

6.16.2.38 `Quadrangle* Physics::getLeftHalfPtr () [inline]`

returns pointer to left half of field

6.16.2.39 `Quadrangle Physics::getLowerHalf () const [inline]`

returns lower half of field

6.16.2.40 `Quadrangle* Physics::getLowerHalfPtr () [inline]`

returns pointer to lower half of field

6.16.2.41 `int Physics::getNumberOfAgents () const [inline]`

returns number of agents

6.16.2.42 `int Physics::getNumberOfEnemies () const [inline]`

returns number of enemies

6.16.2.43 `int Physics::getNumberOfPlayers () const [inline]`

returns number of robots in total

6.16.2.44 `Quadrangle Physics::getObstacleCornerBottomLeft () const [inline]`

returns bottom left corner with collision avoidance margin

6.16.2.45 `Quadrangle* Physics::getObstacleCornerBottomLeftPtr () [inline]`

returns pointer to bottom left corner with collision avoidance margin

6.16.2.46 `Quadrangle Physics::getObstacleCornerBottomRight () const [inline]`

returns bottom right corner with collision avoidance margin

6.16.2.47 `Quadrangle* Physics::getObstacleCornerBottomRightPtr () [inline]`

returns pointer to bottom right corner with collision avoidance margin

6.16.2.48 `Quadrangle Physics::getObstacleCornerTopLeft () const [inline]`

returns top left corner with collision avoidance margin

6.16.2.49 `Quadrangle* Physics::getObstacleCornerTopLeftPtr () [inline]`

returns pointer to top left corner with collision avoidance margin

6.16.2.50 `Quadrangle Physics::getObstacleCornerTopRight () const [inline]`

returns top right corner with collision avoidance margin

6.16.2.51 `Quadrangle* Physics::getObstacleCornerTopRightPtr () [inline]`

returns pointer to top right corner with collision avoidance margin

6.16.2.52 `Quadrangle Physics::getObstacleField () const [inline]`

returns game field with collision avoidance margin

6.16.2.53 `Quadrangle* Physics::getObstacleFieldPtr () [inline]`

returns pointer to game field with collision avoidance margin

6.16.2.54 `Quadrangle Physics::getObstaclePenaltyAreaLeft () const [inline]`

returns left penalty area with collision avoidance margin

6.16.2.55 `Quadrangle* Physics::getObstaclePenaltyAreaLeftPtr () [inline]`

returns pointer to left penalty area with collision avoidance margin

6.16.2.56 `Quadrangle Physics::getObstaclePenaltyAreaRight () const [inline]`

returns right penalty area with collision avoidance margin

6.16.2.57 `Quadrangle* Physics::getObstaclePenaltyAreaRightPtr () [inline]`

returns pointer to right penalty area with collision avoidance margin

6.16.2.58 `Quadrangle Physics::getPenaltyAreaLeft () const [inline]`

returns left penalty area

6.16.2.59 `Quadrangle Physics::getPenaltyAreaRight () const [inline]`

returns right penalty area

6.16.2.60 `Position Physics::getPredBallPosition (int milliseconds) const`

returns predicted ball position

Parameters

<i>milliseconds</i>	prediction time
---------------------	-----------------

6.16.2.61 **Vector2d Physics::getPredBallVelocity** (int *milliseconds*) const

returns predicted ball velocity

Parameters

<i>milliseconds</i>	prediction time
---------------------	-----------------

6.16.2.62 `Quadrangle Physics::getRightHalf () const` `[inline]`

returns right half of field

6.16.2.63 `Quadrangle* Physics::getRightHalfPtr ()` `[inline]`

returns pointer to right half of field

6.16.2.64 `Quadrangle Physics::getUpperHalf () const` `[inline]`

returns upper half of field

6.16.2.65 `Quadrangle* Physics::getUpperHalfPtr ()` `[inline]`

returns pointer to upper half of field

6.16.2.66 `void Physics::initializePhysics ()`

initializes obstacles

6.16.2.67 `bool Physics::isInsideGameField (Position const & pos) const` `[inline]`

checks if position is inside field

Parameters

<i>pos</i>	position to be checked
------------	------------------------

6.16.2.68 `bool Physics::isInsideLeftHalf (Position const & pos) const` `[inline]`

checks if position is inside left half

Parameters

<i>pos</i>	position to be checked
------------	------------------------

6.16.2.69 `bool Physics::isInsideLeftPenaltyArea (Position const & pos) const` `[inline]`

checks if position is inside left penalty area

Parameters

<i>pos</i>	position to be checked
------------	------------------------

6.16.2.70 `bool Physics::isInsideLowerHalf (Position const & pos) const` `[inline]`

checks if position is inside lower half

Parameters

<i>pos</i>	position to be checked
------------	------------------------

6.16.2.71 `bool Physics::isInsidePenaltyArea (Position const & pos) const` `[inline]`

checks if position is inside any penalty area

Parameters

<i>pos</i>	position to be checked
------------	------------------------

6.16.2.72 `bool Physics::isInsideRightHalf (Position const & pos) const` `[inline]`

checks if position is inside right half

Parameters

<i>pos</i>	position to be checked
------------	------------------------

6.16.2.73 `bool Physics::isInsideRightPenaltyArea (Position const & pos) const` `[inline]`

checks if position is inside right penalty area

Parameters

<i>pos</i>	position to be checked
------------	------------------------

6.16.2.74 `bool Physics::isInsideUpperHalf (Position const & pos) const` `[inline]`

checks if position is inside upper half

Parameters

<i>pos</i>	position to be checked
------------	------------------------

6.16.2.75 `void Physics::run ()` `[virtual]`

main update routine of [Physics](#)

Implements [RunnableObject](#).

6.16.2.76 `void Physics::startComparePrediction (int predictionTime)`

start a prediction comparison to check prediction calculation

6.16.2.77 `void Physics::stopComparePrediction ()` `[inline]`

stop the prediction comparison

6.16.3 Member Data Documentation

6.16.3.1 `const double Physics::BALL_OBSTACLE_RADIUS = BALL_RADIUS + 1.5 * ROBOT_RADIUS`

ball radius with collision avoidance margin

6.16.3.2 `const double Physics::BALL_RADIUS = 0.021335`

physical ball radius

6.16.3.3 `const double Physics::CORNER_MARGIN = 1.1 * ROBOT_RADIUS`

collision avoidance margin for corners

6.16.3.4 `const double Physics::FIELD_MARGIN = 1.2 * ROBOT_RADIUS`

collision avoidance margin for field

6.16.3.5 `const double Physics::PENALTY_AREA_MARGIN = 1.05 * ROBOT_RADIUS`

collision avoidance margin for penalty areas

6.16.3.6 `const double Physics::ROBOT_OBSTACLE_RADIUS = 3.5 * ROBOT_RADIUS`

robot radius with collision avoidance margin

6.16.3.7 `const double Physics::ROBOT_RADIUS = 0.047`

physical robot radius

The documentation for this class was generated from the following files:

- [lib/Physics.h](#)
- [src/Physics.cpp](#)

6.17 PointInfo Struct Reference

A struct storing information about a point on the ball trajectory. Stores Position, Time and Velocity.

```
#include <Trajectory.h>
```

Public Member Functions

- [PointInfo](#) ()
- [PointInfo](#) ([Vector2d](#) point, [Vector2d](#) velocity)
- [PointInfo](#) ([Vector2d](#) point, [Vector2d](#) velocity, int time)

Public Attributes

- [Vector2d](#) [Point](#)
- int [Time](#)
- [Vector2d](#) [Velocity](#)

6.17.1 Detailed Description

A struct storing information about a point on the ball trajectory. Stores Position, Time and Velocity.

6.17.2 Constructor & Destructor Documentation

6.17.2.1 `PointInfo::PointInfo () [inline]`

6.17.2.2 `PointInfo::PointInfo (Vector2d point, Vector2d velocity) [inline]`

6.17.2.3 `PointInfo::PointInfo (Vector2d point, Vector2d velocity, int time) [inline]`

6.17.3 Member Data Documentation

6.17.3.1 `Vector2d PointInfo::Point`

6.17.3.2 `int PointInfo::Time`

6.17.3.3 `Vector2d PointInfo::Velocity`

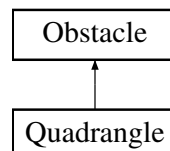
The documentation for this struct was generated from the following file:

- [helper/Trajectory.h](#)

6.18 Quadrangle Struct Reference

```
#include <Quadrangle.h>
```

Inheritance diagram for Quadrangle:



Public Member Functions

- [Quadrangle](#) (const [Quadrangle](#) &quad)
Copy constructor of [Quadrangle](#).
- [Quadrangle](#) (Position pos1, Position pos2, Position pos3, Position pos4)
Constructing a [Quadrangle](#) using 4 Positions. Defined counter clockwise!
- virtual double [getDistance](#) (const Position &pos) const
getter for distance of obstacle to position
- virtual bool [isInside](#) (const Position &pos) const
determines if position is inside an obstacle
- std::vector< [LineSegment](#) > [getSegments](#) () const
getter for line segments of a quadrangle
- virtual std::vector< [Vector2d](#) > [getIntersection](#) (const [Line](#) &line) const
calculates intersection points between a line and an obstacle
- virtual std::vector< [Vector2d](#) > [getIntersection](#) (const [LineSegment](#) &seg) const
calculates intersection points between a line segment and an obstacle

- virtual bool [intersects](#) (const [Line](#) &line) const
determines if there is an intersection between a line and an obstacle
- virtual bool [intersects](#) (const [LineSegment](#) &seg) const
determines if there is an intersection between a line segment and an obstacle
- virtual std::vector< [Vector2d](#) > [getTangentPoints](#) (const Position &pos) const
calculates tangent points of an obstacle
- void [setCenter](#) (const Position &)
- virtual Position [getValidPosition](#) (const Position &pos) const
gets the closest valid position (position outside an obstacle)

Friends

- std::ostream & [operator<<](#) (std::ostream &os, const [Quadrangle](#) &vec)

Additional Inherited Members

6.18.1 Constructor & Destructor Documentation

6.18.1.1 [Quadrangle::Quadrangle](#) (const [Quadrangle](#) &quad)

Copy constructor of [Quadrangle](#).

6.18.1.2 [Quadrangle::Quadrangle](#) (Position *pos1*, Position *pos2*, Position *pos3*, Position *pos4*)

Constructing a [Quadrangle](#) using 4 Positions. Defined counter clockwise!

Parameters

<i>pos1,...,pos4</i>	Position 1 - 4
----------------------	----------------

6.18.2 Member Function Documentation

6.18.2.1 double [Quadrangle::getDistance](#) (const Position &*p*) const [virtual]

getter for distance of obstacle to position

Parameters

<i>p</i>	position
----------	----------

Implements [Obstacle](#).

6.18.2.2 std::vector< [Vector2d](#) > [Quadrangle::getIntersection](#) (const [Line](#) &*line*) const [virtual]

calculates intersection points between a line and an obstacle

Parameters

<i>line</i>	line
-------------	------

Implements [Obstacle](#).

6.18.2.3 std::vector< [Vector2d](#) > [Quadrangle::getIntersection](#) (const [LineSegment](#) &*seg*) const [virtual]

calculates intersection points between a line segment and an obstacle

Parameters

<i>seg</i>	line segment
------------	--------------

Implements [Obstacle](#).

6.18.2.4 `std::vector<LineSegment> Quadrangle::getSegments () const` `[inline]`

getter for line segments of a quadrangle

6.18.2.5 `std::vector< Vector2d > Quadrangle::getTangentPoints (const Position & pos) const` `[virtual]`

calculates tangent points of an obstacle

Parameters

<i>pos</i>	point the tangents shall run through
------------	--------------------------------------

Implements [Obstacle](#).

6.18.2.6 `Position Quadrangle::getValidPosition (const Position & pos) const` `[virtual]`

gets the closest valid position (position outside an obstacle)

Parameters

<i>pos</i>	old position that shall be checked and, if necessary, updated
------------	---

Implements [Obstacle](#).

6.18.2.7 `bool Quadrangle::intersects (const Line & line) const` `[virtual]`

determines if there is an intersection between a line and an obstacle

Parameters

<i>line</i>	line
-------------	------

Implements [Obstacle](#).

6.18.2.8 `bool Quadrangle::intersects (const LineSegment & seg) const` `[virtual]`

determines if there is an intersection between a line segment and an obstacle

Parameters

<i>seg</i>	line segment
------------	--------------

Implements [Obstacle](#).

6.18.2.9 `bool Quadrangle::isInside (const Position & p) const` `[virtual]`

determines if position is inside an obstacle

Parameters

<i>p</i>	position
----------	----------

Implements [Obstacle](#).

6.18.2.10 void `Quadrangle::setCenter (const Position &)` `[inline]`

6.18.3 Friends And Related Function Documentation

6.18.3.1 `std::ostream& operator<< (std::ostream & os, const Quadrangle & vec)` `[friend]`

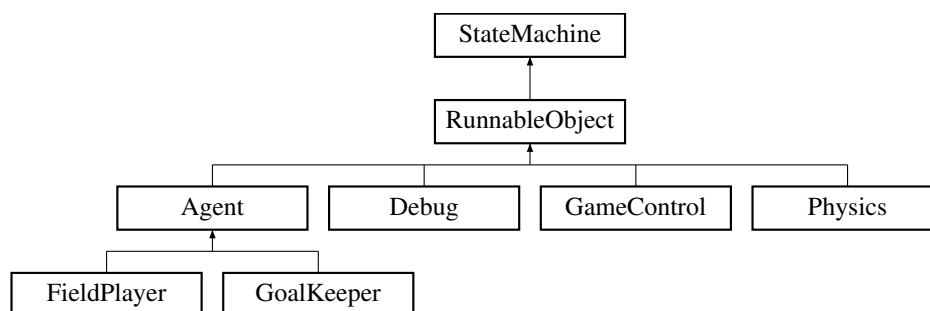
The documentation for this struct was generated from the following files:

- [helper/Quadrangle.h](#)
- [helper/Quadrangle.cpp](#)

6.19 RunnableObject Class Reference

```
#include <RunnableObject.h>
```

Inheritance diagram for RunnableObject:



Public Types

- enum [RunStatus](#) { [RUN](#) = 0, [STOP](#) = 1 }

Public Member Functions

- [RunnableObject](#) (int initState, int interval)
Constructor for the runnable object.
- virtual void [run](#) ()=0
run
- void [stop](#) ()
stop run
- int [getSleepTime](#) ()
calculate sleep time depending on calculation time
- void [restartTimer](#) ()
restart timer_

Protected Attributes

- QTime [timer_](#)
- [RunStatus](#) [status_](#)
- time_t [lastSchedule_](#)
- bool [noWarning_](#) = false

Additional Inherited Members

6.19.1 Member Enumeration Documentation

6.19.1.1 enum RunnableObject::RunStatus

Enumerator

RUN

STOP

6.19.2 Constructor & Destructor Documentation

6.19.2.1 RunnableObject::RunnableObject (int *initState*, int *interval*) [inline]

Constructor for the runnable object.

6.19.3 Member Function Documentation

6.19.3.1 int RunnableObject::getSleepTime () [inline]

calculate sleep time depending on calculation time

6.19.3.2 void RunnableObject::restartTimer () [inline]

restart timer_

6.19.3.3 virtual void RunnableObject::run () [pure virtual]

run

Implemented in [Physics](#), [Agent](#), [GoalKeeper](#), [FieldPlayer](#), [GameControl](#), and [Debug](#).

6.19.3.4 void RunnableObject::stop () [inline]

stop run

6.19.4 Member Data Documentation

6.19.4.1 time_t RunnableObject::lastSchedule_ [protected]

time of last schedule

6.19.4.2 `bool RunnableObject::noWarning_ = false` [protected]

remove warning for one execution of `run()`

6.19.4.3 `RunStatus RunnableObject::status_` [protected]

status of the runnable object

6.19.4.4 `QTime RunnableObject::timer_` [protected]

timer to time

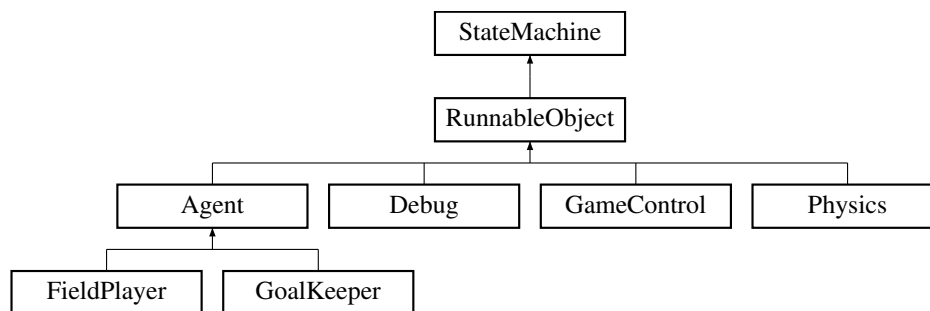
The documentation for this class was generated from the following file:

- `lib/RunnableObject.h`

6.20 StateMachine Class Reference

```
#include <StateMachine.h>
```

Inheritance diagram for StateMachine:



Public Member Functions

- `StateMachine` (int initState, int interval)
Constructor for `StateMachine`.
- `int getState ()`
get the current state of the `StateMachine`
- `void changeState` (int newState)
change the state of the `StateMachine` setting the state change flag and resetting the timer
- `bool after` (int time_in_micros)
returns true if there was no state change in time_in_micros
- `void incTimer ()`
increment timer

Public Attributes

- `int currentState_`
- `int lastState_`
- `bool stateChangeFlag_`

Protected Attributes

- int [interval_](#)
- int [timer_](#)

6.20.1 Constructor & Destructor Documentation

6.20.1.1 `StateMachine::StateMachine (int initState, int interval)` `[inline]`

Constructor for [StateMachine](#).

6.20.2 Member Function Documentation

6.20.2.1 `bool StateMachine::after (int time_in_micros)` `[inline]`

returns true if there was no state change in `time_in_micros`

6.20.2.2 `void StateMachine::changeState (int newState)` `[inline]`

change the state of the [StateMachine](#) setting the state change flag and resetting the timer

6.20.2.3 `int StateMachine::getState ()` `[inline]`

get the current state of the [StateMachine](#)

6.20.2.4 `void StateMachine::incTimer ()` `[inline]`

increment timer

6.20.3 Member Data Documentation

6.20.3.1 `int StateMachine::currentState_`

current state of the [StateMachine](#)

6.20.3.2 `int StateMachine::interval_` `[protected]`

Statemachine interval

6.20.3.3 `int StateMachine::lastState_`

last state of the [StateMachine](#)

6.20.3.4 `bool StateMachine::stateChangeFlag_`

State change flag being set to true for one period

6.20.3.5 int StateMachine::timer_ [protected]

timer counting intervals

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

- lib/[StateMachine.h](#)

6.21 TargetPoint Struct Reference

defines target point and how it is supposed to be reached

```
#include <Path.h>
```

Public Member Functions

- [TargetPoint](#) ()
standard constructor
- [TargetPoint](#) (Position pos, bool robBrake=true)
constructor of target point with position
- [TargetPoint](#) (Position pos, [Vector2d](#) heading, bool robBrake=true)
constructor of target point with position and heading
- [TargetPoint](#) (Position pos, double targetPrecision, bool robBrake=true)
constructor of target point with position and precision
- [TargetPoint](#) (Position pos, [Vector2d](#) heading, double targetPrecision, bool robBrake=true)
constructor of target point with position, heading and precision
- [TargetPoint](#) ([Vector2d](#) pos, bool robBrake=true)
constructor of target point with position
- [TargetPoint](#) ([Vector2d](#) pos, [Vector2d](#) heading, bool robBrake=true)
constructor of target point with position and heading
- [TargetPoint](#) ([Vector2d](#) pos, double targetPrecision, bool robBrake=true)
constructor of target point with position and precision
- [TargetPoint](#) ([Vector2d](#) pos, [Vector2d](#) heading, double targetPrecision, bool robBrake=true)
constructor of target point with position, heading and precision

Public Attributes

- [Vector2d](#) Location
- boost::optional< [Vector2d](#) > Heading
- double Precision = 0.1
- double AnglePrecision = deg2rad(5.)
- bool Brake = true

6.21.1 Detailed Description

defines target point and how it is supposed to be reached

6.21.2 Constructor & Destructor Documentation

6.21.2.1 TargetPoint::TargetPoint () [inline]

standard constructor

6.21.2.2 `TargetPoint::TargetPoint (Position pos, bool robBrake = true)` [inline]

constructor of target point with position

Parameters

<i>pos</i>	target position
<i>robBrake</i>	braking active

6.21.2.3 TargetPoint::TargetPoint (Position *pos*, Vector2d *heading*, bool *robBrake* = true) [inline]

constructor of target point with position and heading

Parameters

<i>pos</i>	target position
<i>heading</i>	target heading
<i>robBrake</i>	braking active

6.21.2.4 TargetPoint::TargetPoint (Position *pos*, double *targetPrecision*, bool *robBrake* = true) [inline]

constructor of target point with position and precision

Parameters

<i>pos</i>	target position
<i>targetPrecision</i>	target precision
<i>robBrake</i>	braking active

6.21.2.5 TargetPoint::TargetPoint (Position *pos*, Vector2d *heading*, double *targetPrecision*, bool *robBrake* = true) [inline]

constructor of target point with position, heading and precision

Parameters

<i>pos</i>	target position
<i>heading</i>	target heading
<i>targetPrecision</i>	target precision
<i>robBrake</i>	braking active

6.21.2.6 TargetPoint::TargetPoint (Vector2d *pos*, bool *robBrake* = true) [inline]

constructor of target point with position

Parameters

<i>pos</i>	target position
<i>robBrake</i>	braking active

6.21.2.7 TargetPoint::TargetPoint (Vector2d *pos*, Vector2d *heading*, bool *robBrake* = true) [inline]

constructor of target point with position and heading

Parameters

<i>pos</i>	target position
<i>heading</i>	target heading
<i>robBrake</i>	braking active

6.21.2.8 `TargetPoint::TargetPoint (Vector2d pos, double targetPrecision, bool robBrake = true) [inline]`

constructor of target point with position and precision

Parameters

<i>pos</i>	target position
<i>targetPrecision</i>	target precision
<i>robBrake</i>	braking active

6.21.2.9 `TargetPoint::TargetPoint (Vector2d pos, Vector2d heading, double targetPrecision, bool robBrake = true) [inline]`

constructor of target point with position, heading and precision

Parameters

<i>pos</i>	target position
<i>heading</i>	target heading
<i>targetPrecision</i>	target precision
<i>robBrake</i>	braking active

6.21.3 Member Data Documentation

6.21.3.1 `double TargetPoint::AnglePrecision = deg2rad(5.)`

angle precision

6.21.3.2 `bool TargetPoint::Brake = true`

status if braking is activated

6.21.3.3 `boost::optional<Vector2d> TargetPoint::Heading`

optional heading direction

6.21.3.4 `Vector2d TargetPoint::Location`

position of target point

6.21.3.5 `double TargetPoint::Precision = 0.1`

precision

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

- lib/[Path.h](#)

6.22 Trajectory Class Reference

```
#include <Trajectory.h>
```

Public Member Functions

- [Trajectory](#) ([Physics](#) *physics)
Constructor for [Trajectory](#) with a pointer to a physics instance.
- void [updateTrajectory](#) ()
updates the whole ball trajectory
- [Vector2d](#) [getPredictedBallPosition](#) (int millis) const
retrieves the predicted ball position after millis
- [Vector2d](#) [getPredictedBallVelocity](#) (int millis) const
retrieves the predicted ball velocity after millis
- [PointInfo](#) [getPredictedPointInfo](#) (int millis) const
retrieves the predicted ball [PointInfo](#) after millis
- std::vector< [PointInfo](#) > [getBallTrajectory](#) () const
retrieves the whole ball [Trajectory](#) as calculated in [updateTrajectory\(\)](#)
- void [printTrajectory](#) ()
print the whole ball trajectory

6.22.1 Constructor & Destructor Documentation

6.22.1.1 [Trajectory::Trajectory](#) ([Physics](#) * *physics*) [inline]

Constructor for [Trajectory](#) with a pointer to a physics instance.

Parameters

<i>physics</i>	Pointer to physics instance
----------------	-----------------------------

6.22.2 Member Function Documentation

6.22.2.1 std::vector< [PointInfo](#) > [Trajectory::getBallTrajectory](#) () const

retrieves the whole ball [Trajectory](#) as calculated in [updateTrajectory\(\)](#)

6.22.2.2 [Vector2d](#) [Trajectory::getPredictedBallPosition](#) (int *millis*) const

retrieves the predicted ball position after millis

Parameters

<i>millis</i>	Milliseconds in the future
---------------	----------------------------

6.22.2.3 [Vector2d](#) [Trajectory::getPredictedBallVelocity](#) (int *millis*) const

retrieves the predicted ball velocity after millis

Parameters

<i>millis</i>	Milliseconds in the future
---------------	----------------------------

6.22.2.4 **PointInfo** Trajectory::getPredictedPointInfo (int *millis*) const

retrieves the predicted ball [PointInfo](#) after *millis*

Parameters

<i>millis</i>	Milliseconds in the future
---------------	----------------------------

6.22.2.5 void Trajectory::printTrajectory ()

print the whole ball trajectory

6.22.2.6 void Trajectory::updateTrajectory ()

updates the whole ball trajectory

The documentation for this class was generated from the following files:

- [helper/Trajectory.h](#)
- [helper/Trajectory.cpp](#)

6.23 **Vector2d** Class Reference

```
#include <Vector2d.h>
```

Public Member Functions

- [Vector2d](#) ()
Default constructor of [Vector2d](#).
- [Vector2d](#) (const [Vector2d](#) &vec)
Copy constructor of [Vector2d](#).
- [Vector2d](#) (double newX, double newY)
Constructing a Vector using coordinates.
- [Vector2d](#) (Position const &pos)
Constructing a Vector using a position.
- [Vector2d](#) (double angle)
Constructing a normalized Vector using an angle.
- double [getLength](#) () const
Calculates length of the Vector.
- double [getLengthSquared](#) () const
Calculates length squared of the Vector.
- double [getDistance](#) (const [Vector2d](#) &vec) const
Calculates distance to another Vector.
- double [getDistance](#) (const Position &pos) const
Calculates distance to a Position.
- double [getAngle](#) (const [Vector2d](#) &vec) const
Calculates angle between this Vector and another one.

- double [getAngle](#) () const
Calculates angle of this Vector.
- void [normalize](#) ()
Normalizes the Vector.
- void [turn](#) (double rad)
Turns the Vector by an angle [rad].
- [Vector2d](#) [getNormalized](#) () const
Calculates normalized Vector in direction of this Vector.
- [Vector2d](#) [getTurned](#) (double rad) const
Calculates turned Vector in mathematical positive direction.
- Position [toPosition](#) () const
Converts the Vector to a Position.
- [Vector2d](#) [operator+](#) ([Vector2d](#) const &rhs) const
Overwrites the + Operator.
- [Vector2d](#) [operator-](#) ([Vector2d](#) const &rhs) const
Overwrites the - Operator.
- [Vector2d](#) [operator*](#) (double times) const
*Overwrites the * Operator with a double value.*
- [Vector2d](#) [operator/](#) (double divide) const
Overwrites the / Operator with a double value.
- double [operator*](#) ([Vector2d](#) const &rhs) const
*Overwrites the * Operator with another Vector.*
- double & [operator\[\]](#) (unsigned int index)
- double [operator\[\]](#) (unsigned int index) const

Public Attributes

- union {
 struct {
 double [x](#)
 double [y](#)
 }
 double [d](#) [2]
};

Friends

- std::ostream & [operator<<](#) (std::ostream &os, const [Vector2d](#) &vec)
- std::ostream & [operator<<](#) (std::ostream &os, const std::vector< [Vector2d](#) > &vec)

6.23.1 Constructor & Destructor Documentation

6.23.1.1 [Vector2d::Vector2d](#) ()

Default constructor of [Vector2d](#).

6.23.1.2 [Vector2d::Vector2d](#) (const [Vector2d](#) & vec)

Copy constructor of [Vector2d](#).

6.23.1.3 Vector2d::Vector2d (double *newX*, double *newY*)

Constructing a Vector using coordinates.

Parameters

<i>newX</i>	X coordinate of Vector
<i>newY</i>	Y coordinate of Vector

6.23.1.4 Vector2d::Vector2d (Position const & *pos*)

Constructing a Vector using a position.

Parameters

<i>pos</i>	Position to be turned to a Vector
------------	-----------------------------------

6.23.1.5 Vector2d::Vector2d (double *angle*)

Constructing a normalized Vector using an angle.

Parameters

<i>angle</i>	angle of the new Vector
--------------	-------------------------

6.23.2 Member Function Documentation

6.23.2.1 double Vector2d::getAngle (const Vector2d & *vec*) const

Calculates angle between this Vector and another one.

Parameters

<i>vec</i>	Other Vector
------------	--------------

Returns

Angle between Vectors [rad]

6.23.2.2 double Vector2d::getAngle () const

Calculates angle of this Vector.

Returns

Returns angle of the Vector

6.23.2.3 double Vector2d::getDistance (const Vector2d & *vec*) const

Calculates distance to another Vector.

Parameters

<i>vec</i>	Other Vector
------------	--------------

Returns

Distance

6.23.2.4 double Vector2d::getDistance (const Position & *pos*) const

Calculates distance to a Position.

Parameters

<i>pos</i>	Position
------------	----------

Returns

Distance

6.23.2.5 double Vector2d::getLength () const

Calculates length of the Vector.

Returns

Length

6.23.2.6 double Vector2d::getLengthSquared () const

Calculates length squared of the Vector.

Returns

Length Squared

6.23.2.7 Vector2d Vector2d::getNormalized () const

Calculates normalized Vector in direction of this Vector.

Returns

The normalized Vector

6.23.2.8 Vector2d Vector2d::getTurned (double *rad*) const

Calculates turned Vector in mathematical positive direction.

Returns

The turned Vector

6.23.2.9 void Vector2d::normalize ()

Normalizes the Vector.

6.23.2.10 Vector2d Vector2d::operator* (double *times*) const

Overwrites the * Operator with a double value.

Parameters

<i>times</i>	Double value for operation
--------------	----------------------------

Returns

Scaled Vector

6.23.2.11 double Vector2d::operator* (Vector2d const & rhs) const

Overwrites the * Operator with another Vector.

Parameters

<i>rhs</i>	Other Vector for dot-product
------------	------------------------------

Returns

Scalar dot-product

6.23.2.12 Vector2d Vector2d::operator+ (Vector2d const & rhs) const

Overwrites the + Operator.

Parameters

<i>rhs</i>	Vector for operation
------------	----------------------

Returns

Added Vectors

6.23.2.13 Vector2d Vector2d::operator- (Vector2d const & rhs) const

Overwrites the - Operator.

Parameters

<i>rhs</i>	Vector for operation
------------	----------------------

Returns

Subtracted Vectors

6.23.2.14 Vector2d Vector2d::operator/ (double divide) const

Overwrites the / Operator with a double value.

Parameters

<i>divide</i>	Double value for operation
---------------	----------------------------

Returns

Scaled Vector

6.23.2.15 `double & Vector2d::operator[] (unsigned int index)`

6.23.2.16 `double Vector2d::operator[] (unsigned int index) const`

6.23.2.17 `Position Vector2d::toPosition () const`

Converts the Vector to a Position.

Returns

The conversion

6.23.2.18 `void Vector2d::turn (double rad)`

Turns the Vector by an angle [rad].

Parameters

<i>rad</i>	Angle in radiant
------------	------------------

6.23.3 Friends And Related Function Documentation

6.23.3.1 `std::ostream& operator<< (std::ostream & os, const Vector2d & vec)` [friend]

6.23.3.2 `std::ostream& operator<< (std::ostream & os, const std::vector< Vector2d > & vec)` [friend]

6.23.4 Member Data Documentation

6.23.4.1 `union { ... }`

6.23.4.2 `double Vector2d::d[2]`

6.23.4.3 `double Vector2d::x`

6.23.4.4 `double Vector2d::y`

The documentation for this class was generated from the following files:

- [helper/Vector2d.h](#)
- [helper/Vector2d.cpp](#)

Chapter 7

File Documentation

7.1 helper/Circle.cpp File Reference

```
#include "Circle.h"
```

Functions

- `std::ostream & operator<< (std::ostream &os, const Circle &circle)`

7.1.1 Function Documentation

7.1.1.1 `std::ostream& operator<< (std::ostream & os, const Circle & circle)`

7.2 helper/Circle.h File Reference

```
#include "Obstacle.h"  
#include "Vector2d.h"
```

Classes

- class `Circle`

7.3 helper/KdTree.cpp File Reference

```
#include "KdTree.h"
```

7.4 helper/KdTree.h File Reference

```
#include "Node.h"
```

Classes

- class [KdTree](#)

7.5 helper/Line.cpp File Reference

```
#include "Line.h"
#include <iostream>
```

Functions

- `std::ostream & operator<< (std::ostream &os, const Line &line)`

7.5.1 Function Documentation

7.5.1.1 `std::ostream& operator<< (std::ostream & os, const Line & line)`

7.6 helper/Line.h File Reference

```
#include "Vector2d.h"
#include <boost/optional.hpp>
```

Classes

- class [Line](#)

7.7 helper/LineSegment.cpp File Reference

```
#include "LineSegment.h"
#include <math.h>
#include <iostream>
```

Functions

- `std::ostream & operator<< (std::ostream &os, const LineSegment &seg)`

7.7.1 Function Documentation

7.7.1.1 `std::ostream& operator<< (std::ostream & os, const LineSegment & seg)`

7.8 helper/LineSegment.h File Reference

```
#include "Line.h"
#include "Vector2d.h"
```

Classes

- class [LineSegment](#)

7.9 helper/Node.cpp File Reference

```
#include "Node.h"
```

7.10 helper/Node.h File Reference

```
#include "lib/Geometry.h"
```

Classes

- class [Node](#)

7.11 helper/Obstacle.cpp File Reference

```
#include "Obstacle.h"
```

Functions

- `std::ostream & operator<< (std::ostream &os, const Obstacle &obst)`

7.11.1 Function Documentation

7.11.1.1 `std::ostream& operator<< (std::ostream & os, const Obstacle & obst)`

7.12 helper/Obstacle.h File Reference

```
#include "Line.h"  
#include "LineSegment.h"  
#include "Vector2d.h"  
#include <iostream>
```

Classes

- class [Obstacle](#)

7.13 helper/Quadrangle.cpp File Reference

```
#include "Quadrangle.h"
```

Functions

- `std::ostream & operator<< (std::ostream &os, const Quadrangle &quad)`

7.13.1 Function Documentation

7.13.1.1 `std::ostream& operator<< (std::ostream & os, const Quadrangle & quad)`

7.14 helper/Quadrangle.h File Reference

```
#include "Obstacle.h"
#include "LineSegment.h"
#include "Vector2d.h"
#include <vector>
```

Classes

- struct [Quadrangle](#)

7.15 helper/Trajectory.cpp File Reference

```
#include "Trajectory.h"
#include <math.h>
#include "lib/Physics.h"
#include <iomanip>
```

7.16 helper/Trajectory.h File Reference

```
#include "lib/Geometry.h"
#include <boost/thread/mutex.hpp>
```

Classes

- struct [PointInfo](#)
A struct storing information about a point on the ball trajectory. Stores Position, Time and Velocity.
- class [Trajectory](#)

7.17 helper/Vector2d.cpp File Reference

```
#include "Vector2d.h"
#include <math.h>
#include <iostream>
```


Functions

- `std::ostream & operator<< (std::ostream &os, const Vector2d &vec)`
- `std::ostream & operator<< (std::ostream &os, const std::vector< Vector2d > &vec)`
- [Vector2d operator-](#) (const Position &pos1, const Position &pos2)

7.17.1 Function Documentation

7.17.1.1 [Vector2d operator-](#) (const Position & *pos1*, const Position & *pos2*)

7.17.1.2 `std::ostream& operator<< (std::ostream & os, const Vector2d & vec)`

7.17.1.3 `std::ostream& operator<< (std::ostream & os, const std::vector< Vector2d > & vec)`

7.18 helper/Vector2d.h File Reference

```
#include "position.h"
#include <vector>
```

Classes

- class [Vector2d](#)

7.19 lib/Agent.h File Reference

```
#include "robo_control.h"
#include <vector>
#include "Path.h"
#include "RunnableObject.h"
#include <boost/optional.hpp>
#include <boost/thread/lock_guard.hpp>
#include <boost/thread/mutex.hpp>
#include "StateMachine.h"
#include <QTime>
```

Classes

- class [Agent](#)

Namespaces

- [STOPBALL_STATES](#)
- [CLEARBALL_STATES](#)
- [SHOOTBALL_STATES](#)
- [PASSTO_STATES](#)
- [SUPPORT_GK_STATES](#)

Enumerations

- enum [STOPBALL_STATES::StopBallStates](#) {
[STOPBALL_STATES::INIT](#), [STOPBALL_STATES::NOT_MOVING_BALL](#), [STOPBALL_STATES::BEFORE_BALL](#), [STOPBALL_STATES::OVERTAKE_BALL](#),
[STOPBALL_STATES::BLOCK_BALL](#), [STOPBALL_STATES::END](#) }
- enum [CLEARBALL_STATES::ClearBallStates](#) {
[CLEARBALL_STATES::INIT](#), [CLEARBALL_STATES::STOP_BALL](#), [CLEARBALL_STATES::NEAR_GOAL](#),
[CLEARBALL_STATES::CLEAR](#),
[CLEARBALL_STATES::END](#) }
- enum [SHOOTBALL_STATES::ShootBallStates](#) {
[SHOOTBALL_STATES::INIT](#), [SHOOTBALL_STATES::GET_BEHIND_BALL](#), [SHOOTBALL_STATES::GET_ON_BALL_GOAL_LINE](#), [SHOOTBALL_STATES::SHOOT_BALL](#),
[SHOOTBALL_STATES::KICK](#), [SHOOTBALL_STATES::END](#) }
- enum [PASSTO_STATES::PassToStates](#) {
[PASSTO_STATES::INIT](#), [PASSTO_STATES::GET_BEHIND_BALL](#), [PASSTO_STATES::GET_ON_BALL_TARGET_LINE](#), [PASSTO_STATES::PASS_BALL](#),
[PASSTO_STATES::END](#) }
- enum [SUPPORT_GK_STATES::SupportGkStates](#) { [SUPPORT_GK_STATES::SHORTEN_ANGLE](#), [SUPPORT_GK_STATES::BLOCK](#), [SUPPORT_GK_STATES::MOVE_AWAY](#) }

7.20 lib/Ball.h File Reference

```
#include "raw_ball.h"
```

Classes

- class [Ball](#)

7.21 lib/Debug.h File Reference

```
#include "GameControl.h"
#include "RunnableObject.h"
```

Classes

- class [Debug](#)

Namespaces

- [DEBUG_STATES](#)

Enumerations

- enum [DEBUG_STATES::DebugStates](#) {
[DEBUG_STATES::NO_DEBUG](#), [DEBUG_STATES::CRUISE](#), [DEBUG_STATES::INTERCEPT](#), [DEBUG_STATES::PENALTY](#),
[DEBUG_STATES::START](#), [DEBUG_STATES::SHOOT](#), [DEBUG_STATES::PREDICTION](#), [DEBUG_STATES::REFEREE](#) }

7.22 lib/Enemy.h File Reference

```
#include "robo_control.h"
```

Classes

- class [Enemy](#)

7.23 lib/FieldPlayer.h File Reference

```
#include "Agent.h"  
#include "RunnableObject.h"  
#include "Physics.h"
```

Classes

- class [FieldPlayer](#)

Namespaces

- [ATTACKER_STATES](#)
- [DEFENDER_STATES](#)
- [KICKOFF_STATES](#)

Enumerations

- enum [DefenderRole](#) { [DEFEND_FRONT](#), [DEFEND_BACK](#), [DEFEND_ALONE](#) }
- enum [ATTACKER_STATES::AttackerStates](#) { [ATTACKER_STATES::ANTICIPATE](#), [ATTACKER_STATES::SHOOT](#) }
- enum [DEFENDER_STATES::DefenderStates](#) { [DEFENDER_STATES::SUPPORT_GK](#), [DEFENDER_STATES::CLEAR_BALL](#), [DEFENDER_STATES::PASS_BALL](#), [DEFENDER_STATES::SHOOT_ON_GOAL](#), [DEFENDER_STATES::BLOCK_ENEMY](#), [DEFENDER_STATES::MOVE_ASIDE](#) }
- enum [KICKOFF_STATES::ClearBallStates](#) { [KICKOFF_STATES::PREPARE](#), [KICKOFF_STATES::PREPARE_KICKOFF](#), [KICKOFF_STATES::SHOOT](#) }

7.23.1 Enumeration Type Documentation

7.23.1.1 enum [DefenderRole](#)

Enumerator

DEFEND_FRONT

DEFEND_BACK

DEFEND_ALONE

7.24 lib/GameControl.h File Reference

```
#include "referee.h"
#include "Physics.h"
#include "GoalKeeper.h"
#include "FieldPlayer.h"
#include "thread"
```

Classes

- class [GameControl](#)

Namespaces

- [GAMECONTROL_STATES](#)
- [STRATEGIES](#)

Enumerations

- enum [GAMECONTROL_STATES::GameControlStates](#) {
 [GAMECONTROL_STATES::REFEREE_INIT](#), [GAMECONTROL_STATES::BEFORE_KICK_OFF](#), [GAMECONTROL_STATES::KICK_OFF](#), [GAMECONTROL_STATES::BEFORE_PENALTY](#),
 [GAMECONTROL_STATES::PENALTY](#), [GAMECONTROL_STATES::PLAY_ON](#), [GAMECONTROL_STATES::PAUSE](#), [GAMECONTROL_STATES::TIME_OVER](#),
 [GAMECONTROL_STATES::DEBUG_CRUISE](#), [GAMECONTROL_STATES::DEBUG_INTERCEPT](#), [GAMECONTROL_STATES::DEBUG_SHOOT](#), [GAMECONTROL_STATES::DEBUG_PASSTO](#),
 [GAMECONTROL_STATES::ATTACKER_MODE](#), [GAMECONTROL_STATES::DEFENDER_MODE](#) }
• enum [STRATEGIES::Strategies](#) { [STRATEGIES::OFFENSIVE](#), [STRATEGIES::DEFENSIVE](#) }

7.25 lib/Geometry.h File Reference

```
#include "helper/Vector2d.h"
#include "helper/Line.h"
#include "helper/LineSegment.h"
#include "helper/Obstacle.h"
#include "helper/Circle.h"
#include "helper/Quadrangle.h"
```

7.26 lib/GoalKeeper.h File Reference

```
#include "Agent.h"
#include "Physics.h"
#include "RunnableObject.h"
#include "FieldPlayer.h"
```

Classes

- class [GoalKeeper](#)

Namespaces

- [GOALKEEPER_STATES](#)
- [GOALKEEPER_KICK_STATES](#)

Enumerations

- enum [GOALKEEPER_STATES::GoalyState](#) { [GOALKEEPER_STATES::AUTO_HOLD_NOT_ACTIVE](#), [GOALKEEPER_STATES::AUTO_HOLD_ACTIVE](#), [GOALKEEPER_STATES::CLEAR_BALL](#), [GOALKEEPER_STATES::PENALTY](#) }
- enum [GOALKEEPER_KICK_STATES::GkKickState](#) { [GOALKEEPER_KICK_STATES::PREPARE](#), [GOALKEEPER_KICK_STATES::SHOOT](#) }

7.27 lib/Path.h File Reference

```
#include "list"
#include "Geometry.h"
#include "helper/KdTree.h"
#include <boost/optional.hpp>
```

Classes

- struct [TargetPoint](#)
defines target point and how it is supposed to be reached
- class [Path](#)

7.28 lib/Physics.h File Reference

```
#include "Agent.h"
#include "Enemy.h"
#include "Ball.h"
#include "RunnableObject.h"
#include "Geometry.h"
#include "QTime"
#include "helper/Trajectory.h"
#include <boost/smart_ptr/shared_ptr.hpp>
#include <vector>
```

Classes

- class [Physics](#)
- struct [Physics::GameField](#)

Functions

- template<typename T >
void [movingAverage](#) (T &value, T updatedValue, double factor=0.8)
Performs a moving average filter on the value.

- `template<typename T >`
`void constraint (T &val, T minVal, T maxVal)`
Template to constrain a value. Enter min value first!!!

7.28.1 Function Documentation

7.28.1.1 `template<typename T > void constraint (T & val, T minVal, T maxVal)`

Template to constrain a value. Enter min value first!!!

Parameters

<i>val</i>	Value to be constraint
<i>minVal</i>	minimum Value
<i>maxVal</i>	maximum Value

7.28.1.2 `template<typename T > void movingAverage (T & value, T updatedValue, double factor = 0.8)`

Performs a moving average filter on the value.

Parameters

<i>value</i>	Value to be filtered
<i>updatedValue</i>	new Value
<i>factor</i>	Factor for the new Value

7.29 lib/RunnableObject.h File Reference

```
#include <time.h>
#include "StateMachine.h"
#include <iostream>
#include <math.h>
#include <QTime>
```

Classes

- class [RunnableObject](#)

7.30 lib/StateMachine.h File Reference

Classes

- class [StateMachine](#)

Macros

- `#define SM_DURING`
Macros to help with the usage of StateMachines.
- `#define SM_EXIT`
- `#define SM_ENTRY`

- `#define SM_END`
- `#define SUBSM_DURING(a)`
Macros to help with the usage of subStatemachines.
- `#define SUBSM_EXIT(a)`
- `#define SUBSM_ENTRY(a)`
- `#define SUBSM_END(a)`

7.30.1 Macro Definition Documentation

7.30.1.1 `#define SM_DURING`

Value:

```
if (!stateChangeFlag_)\
{\
    incTimer();\
    switch (currentState_)\
    {\
```

Macros to help with the usage of StateMachines.

7.30.1.2 `#define SM_END`

Value:

```
default:\
    break;\
}\
stateChangeFlag_ = false;\
}\
lastState_ = currentState_;
```

7.30.1.3 `#define SM_ENTRY`

Value:

```
default:\
    break;\
}\
switch (currentState_)\
{\
```

7.30.1.4 `#define SM_EXIT`

Value:

```
default:\
    break;\
}\
}\
if (stateChangeFlag_)\
{\
    switch (lastState_)\
    {\
```

7.30.1.5 #define SUBSM_DURING(a)

Value:

```
if (!a.stateChangeFlag_)\
{
    a.incTimer();\
    switch (a.currentState_)\
    {
```

Macros to help with the usage of subStatemachines.

7.30.1.6 #define SUBSM_END(a)

Value:

```
default:\
    break;\
}\
a.stateChangeFlag_ = false;\
}\
a.lastState_ = a.currentState_;
```

7.30.1.7 #define SUBSM_ENTRY(a)

Value:

```
default:\
    break;\
}\
switch (a.currentState_)\
{
```

7.30.1.8 #define SUBSM_EXIT(a)

Value:

```
default:\
    break;\
}\
}\
if (a.stateChangeFlag_)\
{\
    switch (a.lastState_)\
    {
```

7.31 src/Agent.cpp File Reference

```
#include "lib/Agent.h"
#include "lib/Path.h"
#include "lib/Physics.h"
#include "helper/Vector2d.h"
```

7.32 src/Debug.cpp File Reference

```
#include "lib/Debug.h"
#include <iostream>
```


7.33 src/Enemy.cpp File Reference

```
#include "lib/Enemy.h"  
#include "lib/Physics.h"
```

7.34 src/FieldPlayer.cpp File Reference

```
#include "lib/FieldPlayer.h"
```

7.35 src/GameControl.cpp File Reference

```
#include "lib/GameControl.h"  
#include <thread>
```

7.36 src/GoalKeeper.cpp File Reference

```
#include "lib/GoalKeeper.h"
```

7.37 src/main.cpp File Reference

```
#include <time.h>  
#include <iostream>  
#include <thread>  
#include "kogmo_rtdb.hxx"  
#include "robo_control.h"  
#include "lib/GoalKeeper.h"  
#include "lib/FieldPlayer.h"  
#include "lib/GameControl.h"  
#include "lib/Debug.h"  
#include "lib/Ball.h"
```

Functions

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

7.37.1 Function Documentation

7.37.1.1 `int main (int argc, char * argv[])`

Use client number according to your lab_roso_stud account number!

This is necessary in order to assure that there are unique connections to the RTDB.

Establish connection to the RTDB.

The connection to the RTDB is necessary in order to get access to the control and the status of the robots which are both stored in the RTDB.

In the RTDB there are also informations about the ball and the other robot positions.

Create the client name with the unique client number

7.38 src/Path.cpp File Reference

```
#include "lib/Path.h"  
#include "lib/Physics.h"
```

7.39 src/Physics.cpp File Reference

```
#include "lib/Physics.h"  
#include "lib/Agent.h"  
#include "lib/Enemy.h"  
#include "raw_ball.h"  
#include <iomanip>
```

Macros

- `#define PHYSICS_UPDATE_INTERVAL 30003`

7.39.1 Macro Definition Documentation

7.39.1.1 `#define PHYSICS_UPDATE_INTERVAL 30003`

Index

- ~Agent
 - Agent, 19
- ~Enemy
 - Enemy, 31
- ~FieldPlayer
 - FieldPlayer, 32
- ~GoalKeeper
 - GoalKeeper, 38
- ~KdTree
 - KdTree, 40
- ~Node
 - Node, 53
- ~Path
 - Path, 60
- ANTICIPATE
 - ATTACKER_STATES, 9
- ATTACKER_MODE
 - GAMECONTROL_STATES, 11
- ATTACKER_STATES
 - ANTICIPATE, 9
 - SHOOT, 9
- AUTO_HOLD_ACTIVE
 - GOALKEEPER_STATES, 12
- AUTO_HOLD_NOT_ACTIVE
 - GOALKEEPER_STATES, 12
- ATTACKER_STATES, 9
 - AttackerStates, 9
- activate
 - Agent, 19
- activateCA
 - Agent, 19
- active_
 - Agent, 22
- addAgent
 - Physics, 65
- addEnemy
 - Physics, 66
- addTargetPoint
 - Agent, 19
- addTargetPoints
 - Agent, 19
- after
 - StateMachine, 82
- Agent, 15
 - ~Agent, 19
 - activate, 19
 - activateCA, 19
 - active_, 22
 - addTargetPoint, 19
 - addTargetPoints, 19
 - Agent, 18
 - attackerModeActive_, 22
 - avoidPenaltyZone, 19
 - calcShootPosition, 19
 - cruise, 20
 - deactivate, 20
 - deactivateCA, 20
 - Debug, 22
 - defenderModeActive_, 22
 - deleteTargetPoints, 20
 - desiredSpeed_, 23
 - driveBackwards, 23
 - enemyFieldHalf_, 23
 - enemyGoalSegment_, 23
 - enemyPenaltyZone_, 23
 - getBallTargetLine, 20
 - getDriveBackwards, 20
 - getPositionFiltered, 20
 - heading_, 23
 - isAtTarget, 20
 - isBehindBall, 20
 - kd_, 23
 - ki_, 23
 - kickOffActive_, 23
 - kp_, 23
 - lastError, 23
 - lastPosition_, 23
 - lastPositionTimestamp_, 24
 - lastSpeed_, 24
 - leftSide_, 24
 - ourSide_, 24
 - ownCornerBottomLeft_, 24
 - ownCornerBottomRight_, 24
 - ownFieldHalf_, 24
 - ownGoalSegment_, 24
 - ownPenaltyZone_, 24
 - penaltyModeActive_, 24
 - ROBOT_RADIUS, 24
 - rightSide_, 24
 - robotSpeed_, 25
 - run, 21
 - setDesiredSpeed, 21
 - setSide, 21
 - setTargetPoint, 21
 - setTargetPoints, 21
 - shoot, 21
 - shootBall, 21
 - shootBallActive_, 25

- shootBallSM, 25
- speedIntegrated_, 25
- startClearBall, 21
- startPassTo, 22
- startShootBall, 22
- startStopBall, 22
- stopAllActions, 22
- targetPointMutex_, 25
- timerAgent_, 25
- timerUpdate_, 25
- turn, 22
- update, 22
- useCA_, 25
- AnglePrecision
 - TargetPoint, 86
- attackerModeActive_
 - Agent, 22
- AttackerStates
 - ATTACKER_STATES, 9
- avoidPenaltyZone
 - Agent, 19
- axis
 - Node, 53
- BEFORE_BALL
 - STOPBALL_STATES, 13
- BEFORE_KICK_OFF
 - GAMECONTROL_STATES, 11
- BEFORE_PENALTY
 - GAMECONTROL_STATES, 11
- BLOCK
 - SUPPORT_GK_STATES, 14
- BLOCK_BALL
 - STOPBALL_STATES, 13
- BLOCK_ENEMY
 - DEFENDER_STATES, 10
- BALL_OBSTACLE_RADIUS
 - Physics, 74
- BALL_RADIUS
 - Physics, 75
- Ball, 25
 - Ball, 26
 - getLastTimestamp, 26
- Brake
 - TargetPoint, 86
- CLEAR
 - CLEARBALL_STATES, 9
- CLEAR_BALL
 - DEFENDER_STATES, 10
 - GOALKEEPER_STATES, 12
- CLEARBALL_STATES
 - CLEAR, 9
 - END, 9
 - INIT, 9
 - NEAR_GOAL, 9
 - STOP_BALL, 9
- CRUISE
 - DEBUG_STATES, 10
- CAtargetPoints_
 - Path, 61
- CLEARBALL_STATES, 9
 - ClearBallStates, 9
- CORNER_MARGIN
 - Physics, 75
- cachedCAtargetPoints_
 - Path, 61
- calcShootPosition
 - Agent, 19
- center_
 - Obstacle, 59
- changeState
 - StateMachine, 82
- child
 - Node, 53
- Circle, 26
 - Circle, 27
 - getDistance, 27
 - getIntersection, 28
 - getRadius, 28
 - getTangentPoints, 28
 - getValidPosition, 28
 - intersects, 28, 29
 - isInside, 29
 - operator<<, 29
- Circle.cpp
 - operator<<, 97
- ClearBallStates
 - CLEARBALL_STATES, 9
 - KICKOFF_STATES, 12
- compute
 - Path, 60
- constraint
 - Physics.h, 106
- CornerBottomLeft
 - Physics::GameField, 36
- CornerBottomRight
 - Physics::GameField, 36
- CornerTopLeft
 - Physics::GameField, 36
- CornerTopRight
 - Physics::GameField, 36
- cruise
 - Agent, 20
- currentState_
 - StateMachine, 82
- d
 - Vector2d, 95
- DEBUG_CRUISE
 - GAMECONTROL_STATES, 11
- DEBUG_INTERCEPT
 - GAMECONTROL_STATES, 11
- DEBUG_PASSTO
 - GAMECONTROL_STATES, 11
- DEBUG_SHOOT
 - GAMECONTROL_STATES, 11
- DEBUG_STATES

- CRUISE, 10
- INTERCEPT, 10
- NO_DEBUG, 10
- PENALTY, 10
- PREDICTION, 10
- REFEREE, 10
- SHOOT, 10
- START, 10
- DEFEND_ALONE
 - FieldPlayer.h, 103
- DEFEND_BACK
 - FieldPlayer.h, 103
- DEFEND_FRONT
 - FieldPlayer.h, 103
- DEFENDER_MODE
 - GAMECONTROL_STATES, 11
- DEFENDER_STATES
 - BLOCK_ENEMY, 10
 - CLEAR_BALL, 10
 - MOVE_ASIDE, 10
 - PASS_BALL, 10
 - SHOOT_ON_GOAL, 10
 - SUPPORT_GK, 10
- DEFENSIVE
 - STRATEGIES, 14
- DEBUG_STATES, 10
 - DebugStates, 10
- DEFENDER_STATES, 10
 - DefenderStates, 10
- deactivate
 - Agent, 20
- deactivateCA
 - Agent, 20
- Debug, 29
 - Agent, 22
 - Debug, 30
 - GameControl, 30
 - GameControl, 35
 - run, 30
- DebugStates
 - DEBUG_STATES, 10
- defenderModeActive_
 - Agent, 22
- DefenderRole
 - FieldPlayer.h, 103
- DefenderStates
 - DEFENDER_STATES, 10
- deleteTargetPoints
 - Agent, 20
- depth
 - KdTree, 40
 - Node, 53
- desiredSpeed_
 - Agent, 23
- directionVector_
 - Line, 47
- driveBackwards
 - Agent, 23
- END
 - CLEARBALL_STATES, 9
 - PASSTO_STATES, 12
 - SHOOTBALL_STATES, 13
 - STOPBALL_STATES, 13
- Enemy, 30
 - ~Enemy, 31
 - Enemy, 31
 - getDistToBall, 31
- enemyFieldHalf_
 - Agent, 23
- enemyGoalSegment_
 - Agent, 23
- enemyPenaltyZone_
 - Agent, 23
- FIELD_MARGIN
 - Physics, 75
- farthestChild
 - Node, 53
- Field
 - Physics::GameField, 36
- FieldPlayer.h
 - DEFEND_ALONE, 103
 - DEFEND_BACK, 103
 - DEFEND_FRONT, 103
- FieldPlayer, 31
 - ~FieldPlayer, 32
 - FieldPlayer, 32
 - FieldPlayer, 32
 - run, 32
 - setAttacker, 32
 - setDefender, 32
 - setKickoffPreparationTarget, 32
 - setSide, 33
 - startAttackerMode, 33
 - startDefenderMode, 33
 - startKickOff, 33
- FieldPlayer.h
 - DefenderRole, 103
- GAMECONTROL_STATES
 - ATTACKER_MODE, 11
 - BEFORE_KICK_OFF, 11
 - BEFORE_PENALTY, 11
 - DEBUG_CRUISE, 11
 - DEBUG_INTERCEPT, 11
 - DEBUG_PASSTO, 11
 - DEBUG_SHOOT, 11
 - DEFENDER_MODE, 11
 - KICK_OFF, 11
 - PAUSE, 11
 - PENALTY, 11
 - PLAY_ON, 11
 - REFEREE_INIT, 11
 - TIME_OVER, 11
- GET_BEHIND_BALL
 - PASSTO_STATES, 12
 - SHOOTBALL_STATES, 13

- GET_ON_BALL_GOAL_LINE
 - SHOOTBALL_STATES, 13
- GET_ON_BALL_TARGET_LINE
 - PASSTO_STATES, 12
- GOALKEEPER_KICK_STATES
 - PREPARE, 11
 - SHOOT, 11
- GOALKEEPER_STATES
 - AUTO_HOLD_ACTIVE, 12
 - AUTO_HOLD_NOT_ACTIVE, 12
 - CLEAR_BALL, 12
 - PENALTY, 12
- GAMECONTROL_STATES, 10
 - GameControlStates, 11
- GOALKEEPER_STATES, 11
 - GoalyState, 12
- GameControl, 33
 - Debug, 30, 35
 - GameControl, 34
 - GameControl, 34
 - run, 34
 - setFieldPlayer1, 34
 - setFieldPlayer2, 34
 - setGoalKeeper, 34
 - setPermanentStrategy, 35
- GameControlStates
 - GAMECONTROL_STATES, 11
- GameField
 - Physics::GameField, 36
- getAgent
 - Physics, 66
- getAgentObstacles
 - Physics, 66
- getAngle
 - Line, 44
 - Vector2d, 91
- getBall
 - Physics, 66
- getBallLastPosition
 - Physics, 66
- getBallObstacle
 - Physics, 66
- getBallPositionFiltered
 - Physics, 66
- getBallTargetLine
 - Agent, 20
- getBallTrajectory
 - Physics, 66
 - Trajectory, 87
- getBallVelocity
 - Physics, 66
- getCenter
 - Obstacle, 57
- getChildren
 - KdTree, 40
 - Node, 53
- getClosestEnemysDistance
 - Physics, 66
- getClosestEnemysDistanceToBall
 - Physics, 68
- getClosestPoint
 - Line, 44
 - LineSegment, 49
- getCornerBottomLeft
 - Physics, 68
- getCornerBottomLeftPtr
 - Physics, 68
- getCornerBottomRight
 - Physics, 68
- getCornerBottomRightPtr
 - Physics, 68
- getCornerTopLeft
 - Physics, 68
- getCornerTopLeftPtr
 - Physics, 68
- getCornerTopRight
 - Physics, 68
- getCornerTopRightPtr
 - Physics, 68
- getDirectionVector
 - Line, 44
- getDistToBall
 - Enemy, 31
- getDistance
 - Circle, 27
 - Line, 45
 - Obstacle, 57
 - Quadrangle, 77
 - Vector2d, 91
- getDistanceExtended
 - LineSegment, 50
- getDriveBackwards
 - Agent, 20
- getEndPoint
 - LineSegment, 50
- getEndVector
 - LineSegment, 50
- getEnemies
 - Physics, 68
- getEnemy
 - Physics, 68
- getEnemyClosestToBall
 - Physics, 69
- getEnemyObstacles
 - Physics, 69
- getField
 - Physics, 69
- getFieldPtr
 - Physics, 69
- getGoalLeft
 - Physics, 69
- getGoalLeftPtr
 - Physics, 69
- getGoalRight
 - Physics, 69
- getGoalRightPtr
 - Physics, 69

- Physics, 69
- getHalfwayLine
 - Physics, 69
- getHalfwayLinePtr
 - Physics, 69
- getId
 - Path, 61
- getInnerGoalLeft
 - Physics, 69
- getInnerGoalLeftPtr
 - Physics, 69
- getInnerGoalRight
 - Physics, 69
- getInnerGoalRightPtr
 - Physics, 69
- getIntersection
 - Circle, 28
 - Line, 45
 - LineSegment, 50
 - Obstacle, 57
 - Quadrangle, 77
- getLastTimestamp
 - Ball, 26
- getLeftHalf
 - Physics, 69
- getLeftHalfPtr
 - Physics, 69
- getLength
 - LineSegment, 50
 - Vector2d, 93
- getLengthSquared
 - Vector2d, 93
- getLowerHalf
 - Physics, 70
- getLowerHalfPtr
 - Physics, 70
- getMiddlePoint
 - LineSegment, 50
- getMiddleVector
 - LineSegment, 50
- getName
 - Obstacle, 58
- getNormalVector
 - Line, 45
- getNormalized
 - Vector2d, 93
- getNumberOfAgents
 - Physics, 70
- getNumberOfEnemies
 - Physics, 70
- getNumberOfPlayers
 - Physics, 70
- getObstacleCornerBottomLeft
 - Physics, 70
- getObstacleCornerBottomLeftPtr
 - Physics, 70
- getObstacleCornerBottomRight
 - Physics, 70
- getObstacleCornerBottomRightPtr
 - Physics, 70
- getObstacleCornerTopLeft
 - Physics, 70
- getObstacleCornerTopLeftPtr
 - Physics, 70
- getObstacleCornerTopRight
 - Physics, 70
- getObstacleCornerTopRightPtr
 - Physics, 71
- getObstacleField
 - Physics, 71
- getObstacleFieldPtr
 - Physics, 71
- getObstaclePenaltyAreaLeft
 - Physics, 71
- getObstaclePenaltyAreaLeftPtr
 - Physics, 71
- getObstaclePenaltyAreaRight
 - Physics, 71
- getObstaclePenaltyAreaRightPtr
 - Physics, 71
- getPenaltyAreaLeft
 - Physics, 71
- getPenaltyAreaRight
 - Physics, 71
- getPositionFiltered
 - Agent, 20
- getPredBallPosition
 - Physics, 71
- getPredBallVelocity
 - Physics, 71
- getPredictedBallPosition
 - Trajectory, 87
- getPredictedBallVelocity
 - Trajectory, 87
- getPredictedPointInfo
 - Trajectory, 88
- getRadius
 - Circle, 28
- getReflection
 - Line, 45
- getRightHalf
 - Physics, 73
- getRightHalfPtr
 - Physics, 73
- getSegments
 - Quadrangle, 78
- getSleepTime
 - RunnableObject, 80
- getStartPoint
 - LineSegment, 50
- getStartVector
 - LineSegment, 50
- getState
 - StateMachine, 82
- getSupportVector
 - Line, 45

- getTangentPoints
 - Circle, [28](#)
 - Obstacle, [58](#)
 - Quadrangle, [78](#)
- getTurned
 - Vector2d, [93](#)
- getUpperHalf
 - Physics, [73](#)
- getUpperHalfPtr
 - Physics, [73](#)
- getValidPosition
 - Circle, [28](#)
 - Obstacle, [58](#)
 - Quadrangle, [78](#)
- GkKickState
 - GOALKEEPER_KICK_STATES, [11](#)
- GoalKeeper, [37](#)
 - ~GoalKeeper, [38](#)
 - GoalKeeper, [38](#)
 - GoalKeeper, [38](#)
 - run, [38](#)
 - setAttacker, [38](#)
 - setFirstDefender, [38](#)
 - setSecondDefender, [38](#)
 - setSide, [38](#)
 - startGoalKeeper, [38](#)
 - startPenaltyMode, [38](#)
 - stopGoalKeeper, [39](#)
- GoalLeft
 - Physics::GameField, [36](#)
- GoalRight
 - Physics::GameField, [36](#)
- GoalyState
 - GOALKEEPER_STATES, [12](#)
- HalfwayLine
 - Physics::GameField, [36](#)
- Heading
 - TargetPoint, [86](#)
- heading_
 - Agent, [23](#)
- helper/Circle.cpp, [97](#)
- helper/Circle.h, [97](#)
- helper/KdTree.cpp, [97](#)
- helper/KdTree.h, [97](#)
- helper/Line.cpp, [98](#)
- helper/Line.h, [98](#)
- helper/LineSegment.cpp, [98](#)
- helper/LineSegment.h, [98](#)
- helper/Node.cpp, [99](#)
- helper/Node.h, [99](#)
- helper/Obstacle.cpp, [99](#)
- helper/Obstacle.h, [99](#)
- helper/Quadrangle.cpp, [99](#)
- helper/Quadrangle.h, [100](#)
- helper/Trajectory.cpp, [100](#)
- helper/Trajectory.h, [100](#)
- helper/Vector2d.cpp, [100](#)
- helper/Vector2d.h, [101](#)
- INIT
 - CLEARBALL_STATES, [9](#)
 - PASSTO_STATES, [12](#)
 - SHOOTBALL_STATES, [13](#)
 - STOPBALL_STATES, [13](#)
- INTERCEPT
 - DEBUG_STATES, [10](#)
- id_
 - Path, [61](#)
- inObstacle
 - KdTree, [40](#)
 - Node, [53](#)
- incTimer
 - StateMachine, [82](#)
- initializePath
 - Path, [61](#)
- initializePhysics
 - Physics, [73](#)
- InnerGoalLeft
 - Physics::GameField, [36](#)
- InnerGoalRight
 - Physics::GameField, [36](#)
- insert
 - KdTree, [40](#)
- intersects
 - Circle, [28](#), [29](#)
 - Line, [45](#)
 - LineSegment, [50](#), [51](#)
 - Obstacle, [58](#)
 - Quadrangle, [78](#)
- interval_
 - StateMachine, [82](#)
- isAtTarget
 - Agent, [20](#)
- isBehindBall
 - Agent, [20](#)
- isInSegmentArea
 - LineSegment, [51](#)
- isInside
 - Circle, [29](#)
 - Obstacle, [58](#)
 - Quadrangle, [78](#)
- isInsideGameField
 - Physics, [73](#)
- isInsideLeftHalf
 - Physics, [73](#)
- isInsideLeftPenaltyArea
 - Physics, [73](#)
- isInsideLowerHalf
 - Physics, [73](#)
- isInsidePenaltyArea
 - Physics, [74](#)
- isInsideRightHalf
 - Physics, [74](#)
- isInsideRightPenaltyArea
 - Physics, [74](#)
- isInsideUpperHalf
 - Physics, [74](#)

- isLeftOfLine
 - Line, 45
- KICK
 - SHOOTBALL_STATES, 13
- KICK_OFF
 - GAMECONTROL_STATES, 11
- KICKOFF_STATES
 - PREPARE, 12
 - PREPARE_KICKOFF, 12
 - SHOOT, 12
- KICKOFF_STATES, 12
 - ClearBallStates, 12
- kd_
 - Agent, 23
- KdTree, 39
 - ~KdTree, 40
 - depth, 40
 - getChildren, 40
 - inObstacle, 40
 - insert, 40
 - KdTree, 39
 - KdTree, 39
 - nearest, 40
 - nodeCount, 40
 - position, 40
 - previous, 41
 - root, 41
- ki_
 - Agent, 23
- kickOffActive_
 - Agent, 23
- kp_
 - Agent, 23
- lastError
 - Agent, 23
- lastPosition_
 - Agent, 23
- lastPositionTimestamp_
 - Agent, 24
- lastSchedule_
 - RunnableObject, 80
- lastSpeed_
 - Agent, 24
- lastState_
 - StateMachine, 82
- LeftHalf
 - Physics::GameField, 36
- leftSide_
 - Agent, 24
- lib/Agent.h, 101
- lib/Ball.h, 102
- lib/Debug.h, 102
- lib/Enemy.h, 103
- lib/FieldPlayer.h, 103
- lib/GameControl.h, 104
- lib/Geometry.h, 104
- lib/GoalKeeper.h, 104
- lib/Path.h, 105
- lib/Physics.h, 105
- lib/RunnableObject.h, 106
- lib/StateMachine.h, 106
- Line, 41
 - directionVector_, 47
 - getAngle, 44
 - getClosestPoint, 44
 - getDirectionVector, 44
 - getDistance, 45
 - getIntersection, 45
 - getNormalVector, 45
 - getReflection, 45
 - getSupportVector, 45
 - intersects, 45
 - isLeftOfLine, 45
 - Line, 42, 44
 - normalVector_, 47
 - operator<<, 47
 - setDirection, 45
 - setDirectionVector, 45
 - setSupport, 47
 - setSupportVector, 47
 - supportVector_, 47
- Line.cpp
 - operator<<, 98
- LineSegment, 47
 - getClosestPoint, 49
 - getDistanceExtended, 50
 - getEndPoint, 50
 - getEndVector, 50
 - getIntersection, 50
 - getLength, 50
 - getMiddlePoint, 50
 - getMiddleVector, 50
 - getStartPoint, 50
 - getStartVector, 50
 - intersects, 50, 51
 - isInSegmentArea, 51
 - LineSegment, 49
 - LineSegment, 49
 - operator<<, 51
- LineSegment.cpp
 - operator<<, 98
- Location
 - TargetPoint, 86
- LowerHalf
 - Physics::GameField, 36
- MOVE_ASIDE
 - DEFENDER_STATES, 10
- MOVE_AWAY
 - SUPPORT_GK_STATES, 14
- main
 - main.cpp, 109
- main.cpp
 - main, 109
- movingAverage
 - Physics.h, 106

NEAR_GOAL
 CLEARBALL_STATES, 9
 NO_DEBUG
 DEBUG_STATES, 10
 NOT_MOVING_BALL
 STOPBALL_STATES, 13
 name_
 Obstacle, 59
 nearest
 KdTree, 40
 nearestChild
 Node, 53
 nearestChildPointer
 Node, 55
 noWarning_
 RunnableObject, 80
 Node, 51
 ~Node, 53
 axis, 53
 child, 53
 depth, 53
 farthestChild, 53
 getChildren, 53
 inObstacle, 53
 nearestChild, 53
 nearestChildPointer, 55
 Node, 52
 parent, 55
 position, 55
 previous, 55
 nodeCount
 KdTree, 40
 normalVector_
 Line, 47
 normalize
 Vector2d, 93

 OFFENSIVE
 STRATEGIES, 14
 OVERTAKE_BALL
 STOPBALL_STATES, 13
 Obstacle, 55
 center_, 59
 getCenter, 57
 getDistance, 57
 getIntersection, 57
 getName, 58
 getTangentPoints, 58
 getValidPosition, 58
 intersects, 58
 isInside, 58
 name_, 59
 Obstacle, 56, 57
 operator<<, 59
 setCenter, 59
 setName, 59
 Obstacle.cpp
 operator<<, 99
 ObstacleCornerBottomLeft
 Physics::GameField, 36
 ObstacleCornerBottomRight
 Physics::GameField, 36
 ObstacleCornerTopLeft
 Physics::GameField, 36
 ObstacleCornerTopRight
 Physics::GameField, 36
 ObstacleField
 Physics::GameField, 36
 ObstaclePenaltyAreaLeft
 Physics::GameField, 36
 ObstaclePenaltyAreaRight
 Physics::GameField, 36
 obstacles_
 Path, 61
 operator<<
 Circle, 29
 Circle.cpp, 97
 Line, 47
 Line.cpp, 98
 LineSegment, 51
 LineSegment.cpp, 98
 Obstacle, 59
 Obstacle.cpp, 99
 Quadrangle, 79
 Quadrangle.cpp, 100
 Vector2d, 95
 Vector2d.cpp, 101
 operator*
 Vector2d, 93, 94
 operator+
 Vector2d, 94
 operator-
 Vector2d, 94
 Vector2d.cpp, 101
 operator/
 Vector2d, 94
 ourSide_
 Agent, 24
 ownCornerBottomLeft_
 Agent, 24
 ownCornerBottomRight_
 Agent, 24
 ownFieldHalf_
 Agent, 24
 ownGoalSegment_
 Agent, 24
 ownPenaltyZone_
 Agent, 24

 PASS_BALL
 DEFENDER_STATES, 10
 PASSTO_STATES, 12
 PASSTO_STATES
 END, 12
 GET_BEHIND_BALL, 12
 GET_ON_BALL_TARGET_LINE, 12
 INIT, 12
 PASS_BALL, 12

PAUSE
 GAMECONTROL_STATES, 11
 PENALTY
 DEBUG_STATES, 10
 GAMECONTROL_STATES, 11
 GOALKEEPER_STATES, 12
 PLAY_ON
 GAMECONTROL_STATES, 11
 PREDICTION
 DEBUG_STATES, 10
 PREPARE
 GOALKEEPER_KICK_STATES, 11
 KICKOFF_STATES, 12
 PREPARE_KICKOFF
 KICKOFF_STATES, 12
 PASSTO_STATES, 12
 PassToStates, 12
 PENALTY_AREA_MARGIN
 Physics, 75
 parent
 Node, 55
 PassToStates
 PASSTO_STATES, 12
 Path, 59
 ~Path, 60
 CAtargetPoints_, 61
 cachedCAtargetPoints_, 61
 compute, 60
 getId, 61
 id_, 61
 initializePath, 61
 obstacles_, 61
 Path, 60
 physics_, 61
 position_, 61
 targetPoints_, 61
 useGameField_, 61
 PenaltyAreaLeft
 Physics::GameField, 37
 PenaltyAreaRight
 Physics::GameField, 37
 penaltyModeActive_
 Agent, 24
 Physics, 62
 addAgent, 65
 addEnemy, 66
 BALL_OBSTACLE_RADIUS, 74
 BALL_RADIUS, 75
 CORNER_MARGIN, 75
 FIELD_MARGIN, 75
 getAgent, 66
 getAgentObstacles, 66
 getBall, 66
 getBallLastPosition, 66
 getBallObstacle, 66
 getBallPositionFiltered, 66
 getBallTrajectory, 66
 getBallVelocity, 66
 getClosestEnemysDistance, 66
 getClosestEnemysDistanceToBall, 68
 getCornerBottomLeft, 68
 getCornerBottomLeftPtr, 68
 getCornerBottomRight, 68
 getCornerBottomRightPtr, 68
 getCornerTopLeft, 68
 getCornerTopLeftPtr, 68
 getCornerTopRight, 68
 getCornerTopRightPtr, 68
 getEnemies, 68
 getEnemy, 68
 getEnemyClosestToBall, 69
 getEnemyObstacles, 69
 getField, 69
 getFieldPtr, 69
 getGoalLeft, 69
 getGoalLeftPtr, 69
 getGoalRight, 69
 getGoalRightPtr, 69
 getHalfwayLine, 69
 getHalfwayLinePtr, 69
 getInnerGoalLeft, 69
 getInnerGoalLeftPtr, 69
 getInnerGoalRight, 69
 getInnerGoalRightPtr, 69
 getLeftHalf, 69
 getLeftHalfPtr, 69
 getLowerHalf, 70
 getLowerHalfPtr, 70
 getNumberOfAgents, 70
 getNumberOfEnemies, 70
 getNumberOfPlayers, 70
 getObstacleCornerBottomLeft, 70
 getObstacleCornerBottomLeftPtr, 70
 getObstacleCornerBottomRight, 70
 getObstacleCornerBottomRightPtr, 70
 getObstacleCornerTopLeft, 70
 getObstacleCornerTopLeftPtr, 70
 getObstacleCornerTopRight, 70
 getObstacleCornerTopRightPtr, 71
 getObstacleField, 71
 getObstacleFieldPtr, 71
 getObstaclePenaltyAreaLeft, 71
 getObstaclePenaltyAreaLeftPtr, 71
 getObstaclePenaltyAreaRight, 71
 getObstaclePenaltyAreaRightPtr, 71
 getPenaltyAreaLeft, 71
 getPenaltyAreaRight, 71
 getPredBallPosition, 71
 getPredBallVelocity, 71
 getRightHalf, 73
 getRightHalfPtr, 73
 getUpperHalf, 73
 getUpperHalfPtr, 73
 initializePhysics, 73
 isInsideGameField, 73
 isInsideLeftHalf, 73

- isInsideLeftPenaltyArea, 73
- isInsideLowerHalf, 73
- isInsidePenaltyArea, 74
- isInsideRightHalf, 74
- isInsideRightPenaltyArea, 74
- isInsideUpperHalf, 74
- PENALTY_AREA_MARGIN, 75
- Physics, 65
- ROBOT_RADIUS, 75
- run, 74
- startComparePrediction, 74
- stopComparePrediction, 74
- Physics.h
 - constraint, 106
 - movingAverage, 106
- Physics::GameField, 35
 - CornerBottomLeft, 36
 - CornerBottomRight, 36
 - CornerTopLeft, 36
 - CornerTopRight, 36
 - Field, 36
 - GameField, 36
 - GoalLeft, 36
 - GoalRight, 36
 - HalfwayLine, 36
 - InnerGoalLeft, 36
 - InnerGoalRight, 36
 - LeftHalf, 36
 - LowerHalf, 36
 - ObstacleCornerBottomLeft, 36
 - ObstacleCornerBottomRight, 36
 - ObstacleCornerTopLeft, 36
 - ObstacleCornerTopRight, 36
 - ObstacleField, 36
 - ObstaclePenaltyAreaLeft, 36
 - ObstaclePenaltyAreaRight, 36
 - PenaltyAreaLeft, 37
 - PenaltyAreaRight, 37
 - RightHalf, 37
 - UpperHalf, 37
- physics_
 - Path, 61
- Point
 - PointInfo, 76
- PointInfo, 75
 - Point, 76
 - PointInfo, 76
 - PointInfo, 76
 - Time, 76
 - Velocity, 76
- position
 - KdTree, 40
 - Node, 55
- position_
 - Path, 61
- Precision
 - TargetPoint, 86
- previous
 - KdTree, 41
 - Node, 55
- printTrajectory
 - Trajectory, 88
- Quadrangle, 76
 - getDistance, 77
 - getIntersection, 77
 - getSegments, 78
 - getTangentPoints, 78
 - getValidPosition, 78
 - intersects, 78
 - isInside, 78
 - operator<<, 79
 - Quadrangle, 77
 - setCenter, 79
- Quadrangle.cpp
 - operator<<, 100
- REFEREE
 - DEBUG_STATES, 10
- REFEREE_INIT
 - GAMECONTROL_STATES, 11
- RUN
 - RunnableObject, 80
- ROBOT_RADIUS
 - Agent, 24
 - Physics, 75
- restartTimer
 - RunnableObject, 80
- RightHalf
 - Physics::GameField, 37
- rightSide_
 - Agent, 24
- robotSpeed_
 - Agent, 25
- root
 - KdTree, 41
- run
 - Agent, 21
 - Debug, 30
 - FieldPlayer, 32
 - GameControl, 34
 - GoalKeeper, 38
 - Physics, 74
 - RunnableObject, 80
- RunStatus
 - RunnableObject, 80
- RunnableObject
 - RUN, 80
 - STOP, 80
- RunnableObject, 79
 - getSleepTime, 80
 - lastSchedule_, 80
 - noWarning_, 80
 - restartTimer, 80
 - run, 80
 - RunStatus, 80
 - RunnableObject, 80

- RunnableObject, 80
- status_, 81
- stop, 80
- timer_, 81
- SHOOT
 - ATTACKER_STATES, 9
 - DEBUG_STATES, 10
 - GOALKEEPER_KICK_STATES, 11
 - KICKOFF_STATES, 12
- SHOOT_BALL
 - SHOOTBALL_STATES, 13
- SHOOT_ON_GOAL
 - DEFENDER_STATES, 10
- SHOOTBALL_STATES
 - END, 13
 - GET_BEHIND_BALL, 13
 - GET_ON_BALL_GOAL_LINE, 13
 - INIT, 13
 - KICK, 13
 - SHOOT_BALL, 13
- SHORTEN_ANGLE
 - SUPPORT_GK_STATES, 14
- START
 - DEBUG_STATES, 10
- STOP
 - RunnableObject, 80
- STOP_BALL
 - CLEARBALL_STATES, 9
- STOPBALL_STATES
 - BEFORE_BALL, 13
 - BLOCK_BALL, 13
 - END, 13
 - INIT, 13
 - NOT_MOVING_BALL, 13
 - OVERTAKE_BALL, 13
- STRATEGIES
 - DEFENSIVE, 14
 - OFFENSIVE, 14
- SUPPORT_GK
 - DEFENDER_STATES, 10
- SUPPORT_GK_STATES
 - BLOCK, 14
 - MOVE_AWAY, 14
 - SHORTEN_ANGLE, 14
- SHOOTBALL_STATES, 12
 - ShootBallStates, 13
- SM_DURING
 - StateMachine.h, 107
- SM_END
 - StateMachine.h, 107
- SM_ENTRY
 - StateMachine.h, 107
- SM_EXIT
 - StateMachine.h, 107
- STOPBALL_STATES, 13
 - StopBallStates, 13
- STRATEGIES, 13
 - Strategies, 14
- SUBSM_DURING
 - StateMachine.h, 107
- SUBSM_END
 - StateMachine.h, 108
- SUBSM_ENTRY
 - StateMachine.h, 108
- SUBSM_EXIT
 - StateMachine.h, 108
- SUPPORT_GK_STATES, 14
 - SupportGkStates, 14
- setAttacker
 - FieldPlayer, 32
 - GoalKeeper, 38
- setCenter
 - Obstacle, 59
 - Quadrangle, 79
- setDefender
 - FieldPlayer, 32
- setDesiredSpeed
 - Agent, 21
- setDirection
 - Line, 45
- setDirectionVector
 - Line, 45
- setFieldPlayer1
 - GameControl, 34
- setFieldPlayer2
 - GameControl, 34
- setFirstDefender
 - GoalKeeper, 38
- setGoalKeeper
 - GameControl, 34
- setKickoffPreparationTarget
 - FieldPlayer, 32
- setName
 - Obstacle, 59
- setPermanentStrategy
 - GameControl, 35
- setSecondDefender
 - GoalKeeper, 38
- setSide
 - Agent, 21
 - FieldPlayer, 33
 - GoalKeeper, 38
- setSupport
 - Line, 47
- setSupportVector
 - Line, 47
- setTargetPoint
 - Agent, 21
- setTargetPoints
 - Agent, 21
- shoot
 - Agent, 21
- shootBall
 - Agent, 21
- shootBallActive_
 - Agent, 25

- shootBallSM
 - Agent, 25
- ShootBallStates
 - SHOOTBALL_STATES, 13
- speedIntegrated_
 - Agent, 25
- src/Agent.cpp, 108
- src/Debug.cpp, 108
- src/Enemy.cpp, 109
- src/FieldPlayer.cpp, 109
- src/GameControl.cpp, 109
- src/GoalKeeper.cpp, 109
- src/Path.cpp, 110
- src/Physics.cpp, 110
- src/main.cpp, 109
- startAttackerMode
 - FieldPlayer, 33
- startClearBall
 - Agent, 21
- startComparePrediction
 - Physics, 74
- startDefenderMode
 - FieldPlayer, 33
- startGoalKeeper
 - GoalKeeper, 38
- startKickOff
 - FieldPlayer, 33
- startPassTo
 - Agent, 22
- startPenaltyMode
 - GoalKeeper, 38
- startShootBall
 - Agent, 22
- startStopBall
 - Agent, 22
- stateChangeFlag_
 - StateMachine, 82
- StateMachine, 81
 - after, 82
 - changeState, 82
 - currentState_, 82
 - getState, 82
 - incTimer, 82
 - interval_, 82
 - lastState_, 82
 - stateChangeFlag_, 82
 - StateMachine, 82
 - StateMachine, 82
 - timer_, 82
- StateMachine.h
 - SM_DURING, 107
 - SM_END, 107
 - SM_ENTRY, 107
 - SM_EXIT, 107
 - SUBSM_DURING, 107
 - SUBSM_END, 108
 - SUBSM_ENTRY, 108
 - SUBSM_EXIT, 108
- status_
 - RunnableObject, 81
- stop
 - RunnableObject, 80
- stopAllActions
 - Agent, 22
- StopBallStates
 - STOPBALL_STATES, 13
- stopComparePrediction
 - Physics, 74
- stopGoalKeeper
 - GoalKeeper, 39
- Strategies
 - STRATEGIES, 14
- SupportGkStates
 - SUPPORT_GK_STATES, 14
- supportVector_
 - Line, 47
- TIME_OVER
 - GAMECONTROL_STATES, 11
- TargetPoint, 83
 - AnglePrecision, 86
 - Brake, 86
 - Heading, 86
 - Location, 86
 - Precision, 86
 - TargetPoint, 83, 85, 86
 - TargetPoint, 83, 85, 86
- targetPointMutex_
 - Agent, 25
- targetPoints_
 - Path, 61
- Time
 - PointInfo, 76
- timer_
 - RunnableObject, 81
 - StateMachine, 82
- timerAgent_
 - Agent, 25
- timerUpdate_
 - Agent, 25
- toPosition
 - Vector2d, 95
- Trajectory, 87
 - getBallTrajectory, 87
 - getPredictedBallPosition, 87
 - getPredictedBallVelocity, 87
 - getPredictedPointInfo, 88
 - printTrajectory, 88
 - Trajectory, 87
 - updateTrajectory, 88
- turn
 - Agent, 22
 - Vector2d, 95
- update
 - Agent, 22
- updateTrajectory

- Trajectory, [88](#)
- UpperHalf
 - Physics::GameField, [37](#)
- useCA_
 - Agent, [25](#)
- useGameField_
 - Path, [61](#)
- Vector2d, [88](#)
 - d, [95](#)
 - getAngle, [91](#)
 - getDistance, [91](#)
 - getLength, [93](#)
 - getLengthSquared, [93](#)
 - getNormalized, [93](#)
 - getTurned, [93](#)
 - normalize, [93](#)
 - operator<<, [95](#)
 - operator*, [93](#), [94](#)
 - operator+, [94](#)
 - operator-, [94](#)
 - operator/, [94](#)
 - toPosition, [95](#)
 - turn, [95](#)
 - Vector2d, [89](#), [91](#)
 - x, [95](#)
 - y, [95](#)
- Vector2d.cpp
 - operator<<, [101](#)
 - operator-, [101](#)
- Velocity
 - PointInfo, [76](#)
- x
 - Vector2d, [95](#)
- y
 - Vector2d, [95](#)