Adaptive Synchronisation for a RoboCup Agent

Jefferson D. Montgomery and Alan K. Mackworth

Computer Science Department of the University of British Columbia
Vancouver, B.C., Canada, V6T 1Z4
{jdm,mack}@cs.ubc.ca

Abstract. We describe an algorithm that adaptively synchronises an agent with its environment enabling maximal deliberation time and improved action success rates. The method balances its reliance upon noisy evidence with internal representations, making it robust to interaction faults caused by both communication and timing. The notion of action correctness is developed and used to analyse the new method as well as two special cases: Internal and External synchronisation. Action correctness is determined online by a novel action accounting procedure that determines the outcome of commanded actions. In conjunction, these elements provide online analysis of agent activity, action confirmation for model prediction, and a coarse measure of the agent's coherence with the environment that is used to adapt its performance.

1 Introduction

Distributed systems, with relevant coordination, can be more efficient and productive than independent entities. This fact has not escaped those in search of a computational model of intelligence where intelligent, distributed agents also promise more efficient and productive solutions. Accordingly, multi-agent systems have become an important and widely researched topic. For example, robot soccer was presented as a suitable domain for exploring the properties of situated, multi-agent systems [9] and has since been developed into a popular yearly competition between artificial soccer teams [7].

However, distributed systems are subject to problems such as partial failure, incomplete observability, and unreliable communication. Therefore, it is an important property for successful distributed systems to be able to efficiently coordinate between individuals, including the environment. Unified operation — a *synchronised* system — can be achieved using accurate system specification [4] or through sufficient communication [6, 12]. Asynchronous operation can result in irrelevant sensing, inconsistent representations, and delayed or failed action; problems which can affect event the most sophisticated skills and plans [2, 5, 10, 14].

Considerable research has been done in the related areas of clock skew detection and distributed system monitoring [6, 8], particularly given recent interest in streaming multimedia. In the RoboCup Simulation league, however, the communication protocol is too restrictive to support the standard solutions developed

in these efforts. Accordingly, simple solutions have been adopted often entailing unjustified assumptions. In this paper, we describe a new adaptive algorithm for synchronising the agent with its environment that is robust to communication delays, lost messages, and timing faults.

In Section 2, we develop the notion of correct action, which defines criteria for a solution and forms the basis of our evaluation and analysis. Section 3 presents a taxonomy of practical issues which is used to motivate and compare typical synchronisation techniques in Section 4. Sections 5 and 6 present a novel, adaptive synchronisation algorithm based on an online accounting procedure for action commands. Finally, the results of an empirical comparison of the presented techniques are included and discussed in Section 7.

2 Robots that Act Correctly

In general, a *robotic system* is the coupling of a *robot* with its *environment*. Further, a robot can be viewed as a *controller* embedded in a *plant*, through which the robot senses and acts in the environment. A general robotic system is shown in Fig. 1. In the RoboCup Simulation league, two distinct software programs (the *agent* and `SoccerServer`) interact through a lossy communication channel. This can be viewed as a robotic system where the `SoccerServer` is the environment, the communication channel is the plant, and the agent is the controller.

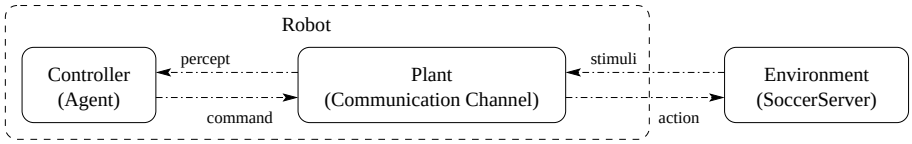


Fig. 1. A general robotic system showing the interactions between the robot's controller and plant as well as with the environment.

The environment effects action and provides stimuli to the robot. `SoccerServer`, shown in Fig. 2, simulates real-life robotic soccer and disseminates stimulus to each of the competing agents [11]. The simulation cycles with a period of 100 milliseconds and can be described by four states. Each cycle begins in the `SB` state (where `sense_body` stimulus describing the agent's condition is provided) and ends in the `E` state (where the domain dynamics are simulated). `SoccerServer` accepts actions in the `R` state and enters the `SE` state every one and a half cycles to provide visual sensor data in the `see` stimulus. Because of the inherent frequency mismatch, state-space is traversed sequentially through three distinct cycle types: type 1 ($\texttt{SB} \rightarrow \texttt{SE} \rightarrow \texttt{R} \rightarrow \texttt{E}$), type 2 ($\texttt{SB} \rightarrow \texttt{R} \rightarrow \texttt{SE} \rightarrow \texttt{R} \rightarrow \texttt{E}$), and type 3 ($\texttt{SB} \rightarrow \texttt{R} \rightarrow \texttt{E}$).

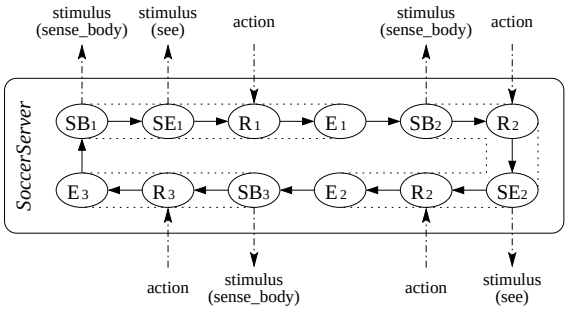


Fig. 2. States and state transitions for the `SoccerServer` environment. To emphasize the three cycle types, state transitions are multiplied out and the cycle type is denoted by a subscript to the state label.

The controller supplies a command trace to the plant to achieve the desired system behaviour. For simplicity, we assume an agent capable of rational thought based on a sense-plan-act architecture, as shown in Fig. 3. This agent receives percepts while in state `S`, then deliberates in state `P`, and finally commands the desired action in state `A`.

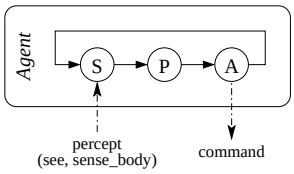


Fig. 3. States and state transitions for the agent.

Successful agents must generate action that is consistent with the current environment. To achieve this coherence, an agent must make decisions based on sensor data and command action in such a way that it is effected while the relevant sensor data is still valid. We call this *correct action*.

Definition 1 (Correct action). In the RoboCup Simulation domain, an agent's actions are *correct* when the following temporal precedence constraints are satisfied, where $\mathtt{X} \prec \mathtt{Y}$ constrains state `X` to happen before state `Y`.

1) $\mathtt{A} \prec \mathtt{E}$
2) $\mathtt{SB} \prec \mathtt{P} \wedge \mathtt{SE} \prec \mathtt{P}$[1]

[1] In cycle type 3, which does not include the `SE` state, the second term of the conjunction is ignored. Otherwise, where $\mathtt{SB} \prec \mathtt{SE}$ is considered a tautology, the first term can be ignored.

The first constraint is strict: `SoccerServer` *must* receive one action command before it begins simulating the dynamics. If $\mathtt{A} \not\prec \mathtt{E}$, the agent will remain idle until the next simulation cycle. If `SoccerServer` receives more than one command in a single cycle, the second and any subsequent command is ignored. Therefore, whether action is effected on time is determined not only by the command reception time, but also by that of the previous cycle as described in Table 1.

Table 1. `SoccerServer` rules for effecting action based on command reception times.

Command reception	Current action
$\mathtt{A}_{prev} \prec \mathtt{E}_{prev} \wedge \mathtt{A} \prec \mathtt{E}$	Effected on time
$\mathtt{A}_{prev} \not\prec \mathtt{E}_{prev} \wedge \mathtt{A} \prec \mathtt{E}$	Failed
$\mathtt{A}_{prev} \prec \mathtt{E}_{prev} \wedge \mathtt{A} \not\prec \mathtt{E}$	Effected late
$\mathtt{A}_{prev} \not\prec \mathtt{E}_{prev} \wedge \mathtt{A} \not\prec \mathtt{E}$	Effected late

The second constraint is soft: the agent *should* consider action only after sufficient sensing. If this constraint is violated then the agent is basing its action on possibly inconsistent sensor data. For example, the CMUnited98 agent acts at the beginning of each simulation cycle [13]. This virtually ensures the first constraint by abandoning the second: all decisions are based on dead-reckoning from the previous cycle and not from current sensor data. Though this method has proven successful in competition where the environmental dynamics are structured and easy to simulate, this approach is ill-advised in general.

3 Desired System Behaviour

Given the required computation for sensor processing, deliberation, and simulation, the constraints in Definition 1 imply that there is an optimal time, t^*, within the simulation cycle when actions should be commanded. If commanding at the optimal time, the agent will receive relevant stimuli, have the maximum amount of time to deliberate, and have all action occur as expected. This ideal behaviour is shown in Fig. 4.

In practice, however, there are several factors that disrupt the ideal behaviour. The simulation cycle is not observable by the agent and appears to fluctuate in its periodicity. These fluctuations are the result of one or more of the following practical considerations which make acting correctly difficult. For example, see Fig. 5.

Clock inaccuracies. The quartz oscillations that computers use to keep time are inaccurate. Typically, a clock can deviate from real time by as much as 30 milliseconds over a 10 minute period [12]. Also, since the system is typically not run over a real-time operating system, interval timers tend to overrun

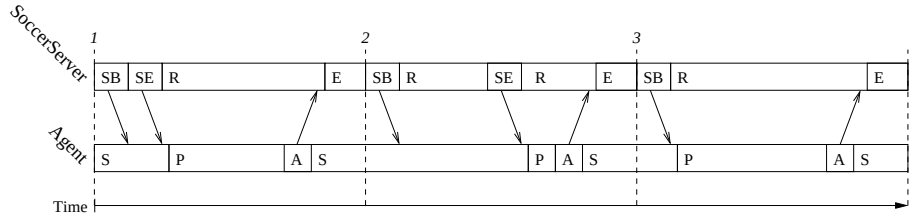


Fig. 4. An event timing diagram depicting ideal, coherent behaviour for each of the three cycle types: all three actions are correct

when the processor is overly taxed. These anomalies occur independently across the system which, over the course of a full game, can incur timing deviations that constitute significant portions of the simulation cycle period.

Varying communication reliability. Varying network traffic causes unobservable and unpredictable communication delays. Further, since the system uses a lossy communication protocol, messages can arrive out of order or be lost entirely in the transport layer. Therefore, the sensitivity of a synchronisation method to such communication faults is extremely relevant.

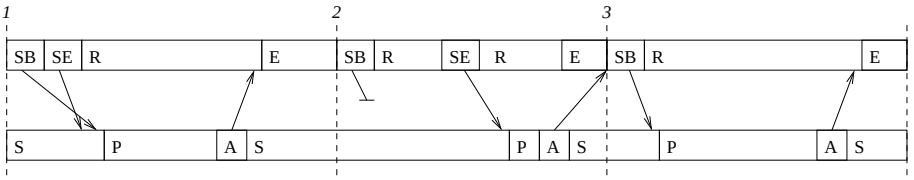


Fig. 5. An example showing possible faults associated with clock inaccuracies and varying network reliability. In the first cycle, the `sense_body` is delayed, arriving out of order with the `see` stimulus. Also, the `SoccerServer`'s interval timer overruns causing the next `sense_body` (which ends up being lost in transit) to be delayed. In the second cycle, the action command is delayed and not effected until the third cycle. Since this action arrives late, the agent remains idle and the action commanded in the third cycle is ignored by `SoccerServer`.

4 Synchronisation Algorithms

In order to synchronise with the environment, each agent must determine its best estimate of the environment's state, which is not directly observable. This section presents two estimation techniques based on internal and external evidence and compares them based on their likelihood of producing correct aciton.

4.1 Synchronisation using Internal Evidence

Internal synchronisation uses internal evidence to coordinate with the environment. The evidence commonly used is the expiration of an interval timer that mimics the simulation periodicity: the agent assumes a new simulation cycle upon the timer's expiry every 100 milliseconds [3]. The agent then commands an action at some fixed *internal* time, $\hat{t^*}$. An example is shown in Fig. 6.

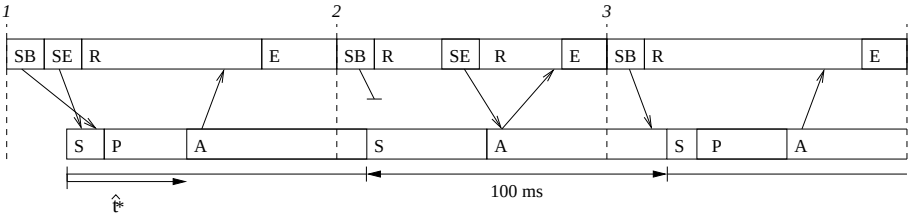
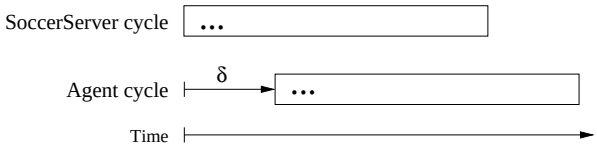


Fig. 6. An example of Internal synchronisation where an internal cycle is initiated upon expiration of an interval timer. The first and third cycles show correct action, although the agent is not fully utilizing the available deliberation time. In the second cycle, the action is scheduled before the **see** stimulus. Therefore, the action decision is based on previous (possibly inconsistent) sensor data.

Due to the internal nature of this algorithm, there are no observed fluctuations due to network variation. However, there will always exist a temporal offset, δ, between the true simulation cycle and the agent's internal representation:



The fundamental assumption behind Internal synchronisation is that the internal and external cycles are in phase (i.e., $\delta = 0$[2]). If this assumption holds the internal method produces excellent results, but assuming correlation is unjustified. In general, δ is uniformly distributed depending on when the agent is initiated and varies dynamically due to clock inaccuracies.

$$P(\delta = t) \equiv \begin{cases} \frac{1}{100} & \text{if } t \in [-50, 50] \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

In order to determine how violation of this fundamental assumption effects action correctness, we consider how likely it is that Definition 1 will hold under

[2] Unless otherwise specified, this paper describes time in integer milliseconds. Units will be omitted for the remainder of the paper.

these conditions. Taking Table 1 into consideration, the probability of correct action is

$$P\left(\text{Correct action}\right) \equiv P\left(\left(\mathtt{A}_{prev} \prec \mathtt{E}_{prev} \wedge \mathtt{A} \prec \mathtt{E}\right) \wedge \left(\mathtt{SB} \prec \mathtt{P} \wedge \mathtt{SE} \prec \mathtt{P}\right)\right) \tag{2}$$

To proceed, we assume that the communication channel introduces a transmission delay, T_{delay}, that is exponentially distributed with expected delay τ, and independent of time [1].

$$P\left(T_{delay} = t\right) \equiv \begin{cases} \frac{1}{\tau} e^{-\frac{t}{\tau}} & \text{if } t \geq 0 \\ 0 & \text{if } t < 0 \end{cases} \tag{3}$$

If T_{see} is the external cycle time that the `see` stimulus is provided (or the `sense_body` stimulus during cycle type 3), then

$$P\left(T_{see} = t\right) \equiv \begin{cases} \frac{2}{3} & \text{if } t = 0 \\ \frac{1}{3} & \text{if } t = 50 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Finally, if T_{delib} and T_{sim} are the time required to decide on and to simulate an action respectively (both of which are assumed constant) then the probability of correct action using Internal synchronisation follows directly from Equation 2:

$$P\left(\begin{array}{c|c} \text{Correct} & \text{Internal} \\ \text{action} & \text{synchronisation} \end{array}\right) = P\left(\begin{array}{l} \left(\delta - 100 + \hat{t^*} + T^{(1)}_{delay} + T_{sim} \leq 0\right) \\ \wedge \left(\delta + \hat{t^*} + T^{(2)}_{delay} + T_{sim} \leq 100\right) \\ \wedge \left(T_{see} + T^{(3)}_{delay} + T_{delib} \leq \delta + \hat{t^*}\right) \end{array}\right) \tag{5}$$

Equation 5 can be computed analytically, and a partial elevation plot is shown in Fig. 7. It shows that Internal synchronisation is not very sensitive to network variation, but that the probability of correct action is low because δ is unlikely to fall in the desired range.

4.2 Synchronisation using External Evidence

As demonstrated in Section 4.1, the probability of correct action is directly related to the validity of the internal representation. External synchronisation abandons the internal timer and uses an external indicator to achieve a more favourable δ distribution. Perception of a `sense_body` stimulus is used as evidence of the `SB` state and of a new `SoccerServer` cycle. As shown in Fig. 8, this evidence triggers a new internal cycle and action commands are scheduled after a fixed internal time, $\hat{t^*}$ $[3, 10]$.

The fundamental assumption behind External synchronisation is that variation in communication delay is negligible and that an appropriate value for the command time, $\hat{t^*}$, is known. Under these conditions, this method produces

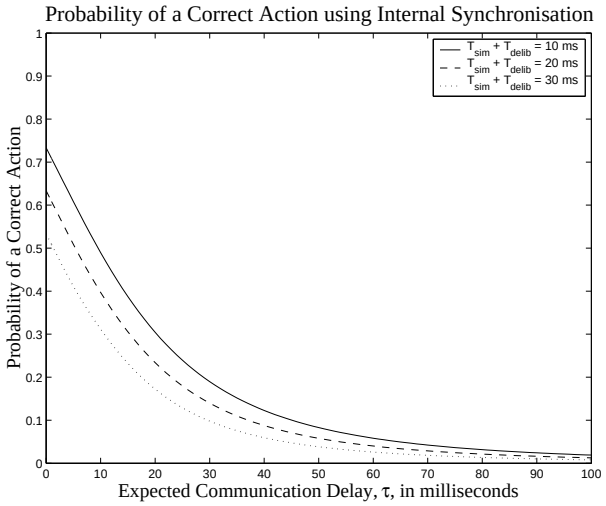


Fig. 7. The probability of correct action using Internal synchronisation.

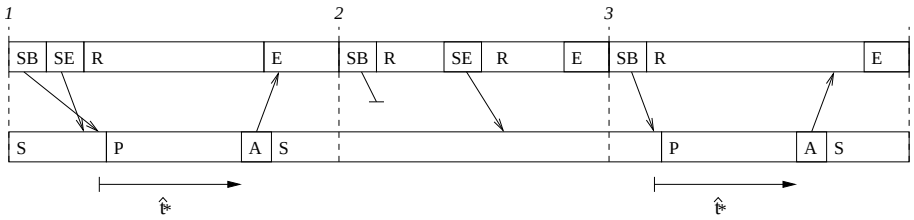


Fig. 8. An example of External synchronisation where action is commanded a fixed time, $\hat{t^*}$, after `sense_body` perception. In the first cycle, the action command is not received in time and the agent remains idle. In the second cycle, the `sense_body` message is lost: no internal cycle is initiated, and no action is commanded. The second action (in the third cycle) is correct.

excellent results. In real systems these assumptions are often violated, however, and reliance upon an external indicator introduces sensitivity to lost stimuli (as shown in Fig. 8). The effect of this assumption on action correctness is shown, continuing from Equation 2, by the corresponding likelihood of correct action in Equation 6 and Fig. 9.

$$P\left(\begin{array}{c}\text{Correct}\\ \text{action}\end{array}\middle|\begin{array}{c}\text{External}\\ \text{synchronisation}\end{array}\right) = P\left(\begin{array}{c}\left(T^{(1)}_{delay} + \hat{t^*} + T^{(2)}_{delay} + T_{sim} \leq 100\right)\\ \wedge\left(T^{(3)}_{delay} + \hat{t^*} + T^{(4)}_{delay} + T_{sim} \leq 100\right)\\ \wedge\left(T_{see} + T^{(5)}_{delay} + T_{delib} \leq T^{(3)}_{delay} + \hat{t^*}\right)\end{array}\right) \quad (6)$$

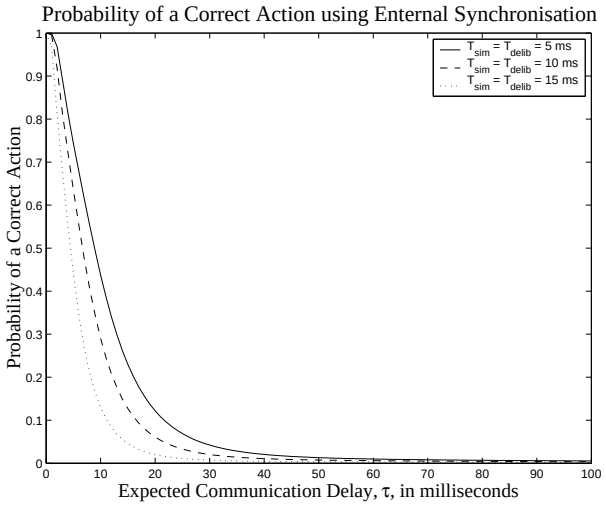


Fig. 9. The probability of correct action using External synchronisation.

If the communication assumptions are valid then the `sense_body` percept is a good indicator of a new simulation cycle. This is clear from the peak in probable correct actions for low τ, but when transmission delay varies the probability decreases rapidly. This is particularly evident with large communication delays where, as shown in Fig. 10, External synchronisation can perform worse than the naive Internal synchronisation.

5 Adaptive Synchronisation

The analysis in Section 4 (specificly Fig. 10) motivates a more robust synchronisation algorithm. Ideally, such an algorithm would combine the reliability of Internal synchronisation with external information to improves the δ distribution.

The proposed *Dynamic Action Index* algorithm uses an interval timer as the internal representation of the simulation cycle, and schedules action commands at the *action index*, I, as shown in Fig. 11. The action index is the agent's estimate of the optimal internal time to command an action, $\widehat{t^* - \delta}$, which is approximated using separate estimates of δ and t^*:

$$I \equiv \hat{t^*} - \hat{\delta} \tag{7}$$

As described in Sections 5.1, 5.2, and Table 3, the Dynamic Action Index algorithm generalises Internal and External synchronisation and relaxes their fundamental assumptions by using improved, adaptive estimates.

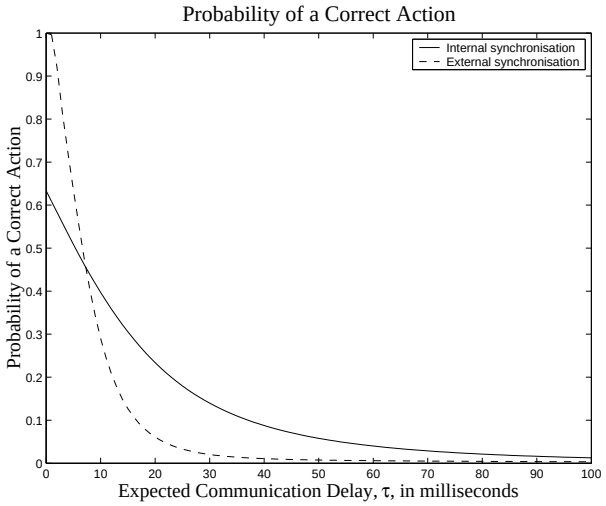


Fig. 10. A comparison of the Internal and External synchronisation algorithms when $T_{sim} + T_{delib} = 10$. As communication delays increase, External synchronisation performance degrades more rapidly. Internal synchronisation becomes favourable when $\tau \tilde{>} 8$.

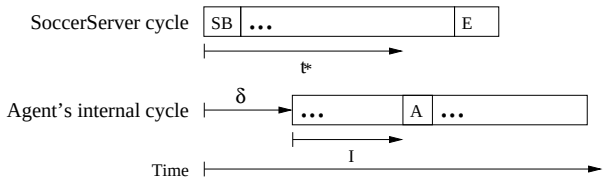


Fig. 11. Synchronisation using a Dynamic Action Index (showing correct estimates), where action is commanded at an internal time, I, which is modified based on its inferred validity.

5.1 Estimating the Internal Cycle Offset

The internal cycle offset, δ, is estimated by modeling noise in `sense_body` perception times. The model is based on the conservative assumption that a quick transmission is likely to occur within a small number of simulation cycles [4, 1]. If we assume that one out of N `sense_body` messages will be delivered promptly, then a good estimate of the cycle offset is given by Equation 8, where $t^{(i)}_{\texttt{sense\_body}}$ is the internal perception time of the ith previous `sense_body`.

$$\hat{\delta} = \min \bigcup_{i=1}^{N} \left\{ t^{(i-1)}_{\texttt{sense\_body}} \right\} \tag{8}$$

Although $\hat{\delta}$ is always an over-estimate (because it includes the transit time of the quickest message), the bias can be neglected by considering necessary communication delays while estimating the optimal time to command action, i.e., by assuming known cycle offset.

5.2 Estimating the Optimal Time to Command Action

Because both network delays and processor strain vary over time, so too does the optimal time, t^*, to command action. Therefore, it is beneficial to adapt the estimate of the optimal command time, $\hat{t^*}$, during operation. The validity of the current estimate can be inferred if the agent can confirm how action is being effected. This inference is summarised in Table 2 and because it operates on the timescale of cycles (where communication delay can be deemed negligible) and does not depend on `sense_body` perception times, reliance upon network responsiveness is minimal.

Table 2. Inferring the validity of the optimal command time estimate assuming known cycle offset.

Action	Cause	Inference
Effected on time	$I \leq t^* - \delta$	$\hat{t^*} \leq t^*$
Effected late	$I > t^* - \delta$	$\hat{t^*} > t^*$
Failed	$(I + \delta) \mod 100 \simeq 0$	$\hat{t^*} \mod 100 \simeq 0$

If the action was effected on time, then the action index was not too large. However, if the estimate is less than optimal then it can be increased (thereby increasing sensing and deliberation) while still commanding action on time. Because the optimal time is not observable, there is little indication of how much to increase the estimate where there is risk of commanding future actions late.

If the action was effected late, then the action index was too large and should be decreased. Because the true amount is unknown, the estimate is decreased by some fixed amount for each late action.

If the action failed, then the command was either lost in transport (which has no implication on synchronisation), or it was ignored by `SoccerServer`. The latter case occurs often if the estimate is on the simulation cycle boundary (in which case the estimate can be reset to its initial value) or intermittently when a single command is excessively delayed (which does not imply an incorrect estimate).

Each inference case can be adversely affected by periodic communication delays. This side effect can be avoided by assuming that each delay is independent and identically distributed and by requiring successive evidence of any inference.

Table 3. The internal cycle offset and optimal command time estimates used in the Internal, External, and Dynamic Action Index synchronisation algorithms.

Algorithm	Estimates	
	Internal cycle offset ($\hat{\delta}$)	Optimal command time ($\hat{t^*}$)
Internal	0	constant (e.g., 70)
External	$t^0_{\texttt{sense\_body}}$	constant (e.g., 70)
Dynamic Action Index	$\min \bigcup_{i=0}^{N-1} \left\{ t^i_{\texttt{sense\_body}} \right\}$	inferred and modified online

6 Confirming Commanded Action

The validity of the optimal command time estimate is inferred based on the outcome of commanded action. Each action is classified by assiging it to one of two disjoint sets: P is an ordered set of pending action, and C is a set of confirmed action. Actions initially belong to P and only become members of C if they obey a specific causal relationship between time and the count of effected actions, n_e.[3]

Definition 2 (Action age). The *age* of a commanded action is the number of `sense_body`'s perceived while waiting for its confirmation.

Definition 3 (Confirmed action). The ith pending action is *confirmed*, if $n_e \geq \|C\| + i$.

If a pending action of age 1 is confirmed, that action was effected on time. Otherwise, it can be difficult to distinguish between late and failed action. The distinction can be made by imposing a bound, τ_{max}, on communication delays, above which unaccounted messages are assumed lost. The confirmation relationship is made explicit in Table 4.

Table 4. Inferring the outcome of an unambiguous action from the count of effected actions.

Count of effected actions	ith pending action	
	Age	Confirmed outcome
$n_e \geq i$	1	Effected on time
$n_e \geq i$	$\left[2, 1 + 2\left\lceil \frac{\tau_{max}}{100} \right\rceil\right)$	Effected late
$n_e < i$	$\left[1 + 2\left\lceil \frac{\tau_{max}}{100} \right\rceil, \infty\right)$	Failed

[3] A count of effected actions is included as part of the `sense_body` stimulus. Any two stimuli can be used to determine the number of actions effected between them.

However, ambiguities arise if a new action is commanded before a previous action has been confirmed, which often occurs while waiting to distinguish between late and failed action. The ith pending action is *ambiguous* if $i < \|P\|$. In such circumstances, all hypotheses are maintained until the ambiguity can be resolved or can be proven to be undecidable.[4]

The basis of the disambiguation procedure are the strict rules that govern how an action command is handled by `SoccerServer`, based on when it is received (see Table 1). In particular, we exploit the fact that an action cannot be effected on time following late action. The procedure is outlined in Table 5.

Table 5. A procedure for disambiguating the set of pending actions, $P = \{p_1, ...\}$ based on the property that late-on time action pairs can not exists in P.

`if` p_i effected on time `then`
    $\forall j \leq i$ action p_j effected on time
`else if` p_i effected late $\vee n_e = \|C \cup P\|$ `then`
    $\forall j \geq i$ action p_j effected late
`else if` p_i lost `then`
    `if` p_{i-1} effected on time `then`
        $\forall j \leq i-1$ action p_j effected on time
    `else if` p_{i-1} effected late `then`
        `for each` j `in` 1 `up to` $i-2$ `do`
            `if` p_j effected late `then`
                $\forall k \geq j$ action p_k effected late
    `else if` $i = \|P\|$ `then`
        $\forall j \leq i$ action p_j ambiguous, $\|C \cup P\| - n_e$ of which are lost

---

7 Comparing of Synchronisation Algorithms

Synchronisation algorithms can be compared in a number of ways. Butler *et al.* develop a system for automatically extracting team activity as well as idle and failure ratios from game log files [3]. However, these measures do not encode the notion of action correctness (for example, late and failed actions increase activity). The online procedure described in Section 6 provides information that is more expressive. In fact, the measures used in [3] can be directly calculated

[4] Ambiguities are only undecidable when a failed action cannot be assigned to a specific pending command. For the purpose of $\hat{t^*}$ estimation, this does not matter because failure is certain, but this does effect model prediction and activity analysis since the agent does not know which action failed.

using counts of correct, late, and failed actions. The ability is a special feature of the online algorithm, which combines the intended action time with inferred status.

The three synchronisation methods described in this paper were tested in competition against previous RoboCup competitors [15]. All algorithms were implemented using the Dynamic Action Index framework (i.e., action commands were scheduled $\hat{t}^* - \hat{\delta}$ after internal cycle initiation) using the parameters specified in Table 3. The status of commanded actions was recorded throughout # partial games played using a network that introduced varying transmission delays of approximately 20 ± 15 milliseconds. The fraction of correct action varied with synchronisation algorithm as shown in Fig. 12.

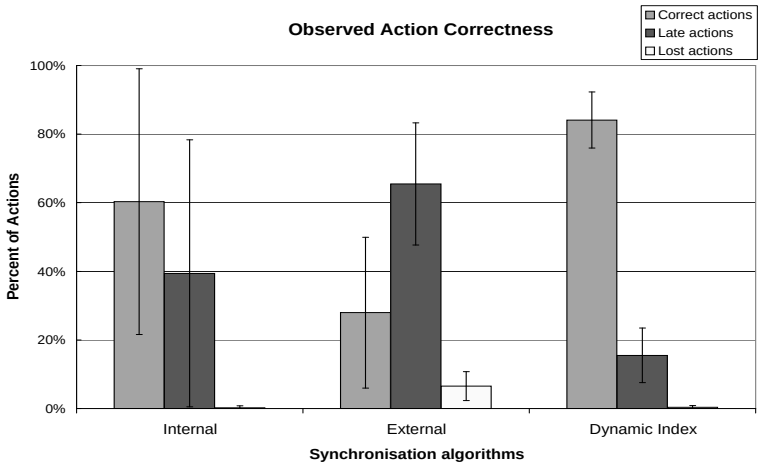


Fig. 12. Experimentally observed distributions of correct, late, and failed action for agents using the Internal, External, and Dynamic Action Index synchronisation algorithms.

The higher variance observed using Internal synchronisation is significant. It corresponds to the uniform distribution of possible cycle offsets. If the internal cycle happens to be synchronised with the external cycle the agent is able to perform a high fraction of correct action. However, poor synchronisation is just as likely.

External synchronisation was observed to produce fewer correct actions than the other algorithms on average. For External synchronisation, variance from the

average is a result of sensitivity to `sense_body` perception times: small changes in communication delay can cause a large change in the fraction of actions that are correct.

The Dynamic Action Index synchronisation algorithm produces the most correct actions on average. Also, the observed variance is low because the δ and t^* estimates continually converge on optimal values making the algorithm robust to varying conditions.

8 Conclusion

This paper introduces the Dynamic Action Index synchronisation algorithm that estimates internal cycle offset and optimal action command times online. We have shown that the Internal and External synchronisation algorithms are each extremal special cases of the new algorithm, providing a satisfying theoretical framework. Moreover, our experimental results show that our new algorithm outperforms the other two approaches under varying network delays. We have shown that combining an internal synchronisation algorithm with estimates derived from the external `SoccerServer` performs better than either alone. In general, any agent must weight and integrate its internal representation with external evidence for optimal performance. Our results demonstrate how this can be achieved for even the low-level task of synchronisation. Practically, agents using the algorithm can choose the optimal time to act and make good decisions, approaching the rational ideal.

References

1. Jean-Chrysostome Bolot. Characterizing End-to-End Packet Delay and Loss in the Internet. *Journal of High-Speed Networks* **2**(3):305–323, December 1993.
2. Hans-Dieter Burkhard, Joscha Bach, Kay Schrter, Jan Wendler , Michael Gollin, Thomas Meinert, and Gerd Sander. AT Humboldt 2000 (Team Description). In P. Stone, T. Balch, and G. Kraetzschmar, editors, RoboCup 2000: Robot Soccer. World Cup IV (LNAI No. 2019), pages 405–408, Springer-Verlag, Berlin, 2000.
3. Marc Butler, Mikhail Prokopenko, and Thomas Howard. Flexible Synchronisation within RoboCup Environment: a Comparative Analysis. In P. Stone, T. Balch, and G. Kraetzschmar, editors, RoboCup 2000: Robot Soccer. World Cup IV (LNAI No. 2019), pages 119–128, Springer-Verlag, Berlin, 2000.
4. F. Cristian. Probabilistic Clock Synchronization. *Distributed Computing* 3:146–158. 1989.
5. Klaus Dorer. Improved Agents of the magmaFreiburg2000 Team. In P. Stone, T. Balch, and G. Kraetzschmar, editors, RoboCup 2000: Robot Soccer. World Cup IV (LNAI No. 2019), pages 417–420, Springer-Verlag, Berlin, 2000.
6. Orion Hodson, Colin Perkins, and Vicky Hardman. Skew Detection and Compensation for Internet Audio Applications. Proceedings of the IEEE International Conference on Multimedia and Expo, New York, NY, July 2000.
7. Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda and Eiichi Osawa. RoboCup: The Robot World Cup Initiative. In W. Lewis Johnson and Barbara

Hayes-Roth, editors, Proceedings of the First International Conference on Autonomous Agents (Agents'97), pages 340–347. ACM Press, New York, 1997.
8. J. L. Lamport. Time, Clocks and the Ordering of Events in a Distributed System. *Communications of the ACM* **21**(7):558–565, 1978.
9. Alan K. Mackworth. On Seeing Robots. In A. Basu and X. Li, editors, Computer Vision: Systems, Theory, and Applications, pages 1–13, World Scientific Press, Singapore, 1993.
10. Birgit Schappel and Frank Schulz. Mainz Rolling Brains 2000. In P. Stone, T. Balch, and G. Kraetzschmar, editors, RoboCup 2000: Robot Soccer. World Cup IV (LNAI No. 2019), pages 497–500, Springer-Verlag, Berlin, 2000.
11. Itsuki Noda. Soccer Server: a Simulator for RoboCup. JSAI AI-Symposium 95: Special Session on RoboCup, Dec. 1995
12. Rafail Ostrovsky and Boaz Patt-Shamir. Optimal and Efficient Clock Synchronization Under Drifting Clocks. In Proceedings of ACM Symposium on PODC '99. Atlanta, GA, 1999.
13. Peter Stone, Manuela Veloso, and Patrick Riley. The CMUnited-98 Champion Simulator Team. (extended version) In M. Asada and H. Kitano, editors, RoboCup-98: Robot Soccer World Cup II (LNAI No. 2019), Springer-Verlag, Berlin, 2000.
14. Shahriar Pourazin. Pardis. In Manuela M. Veloso, Enrico Pagello and Hiroaki Kitano, editors, RoboCup-99: Robot Soccer World Cup III. (LNCS No. 1856), pages 614–617, Springer-Verlag, Berlin, 2000.
15. The RoboCup Simulator Team Repository, `http://medialab.di.unipi.it/Project/Robocup/pub/`, January 31, 2002.